Motion Engineering, Inc.

33 South La Patera Lane
Santa Barbara, CA 93117
ph (805) 681-3300
fax (805) 681-3311
info@motioneng.com

DSP & DSPpro Series Sample Applications Reference

Mar 2002

Sample Applications Reference

Mar 2002
Part # M001-0003 rev B

Motion Engineering, Inc.
33 South La Patera Lane
Santa Barbara, CA 93117-3214
ph 805-681-3300
fax 805-681-3311
e-mail technical@motioneng.com

Motion Engineering, Inc.

1 QUICK REFERENCE

2 CODE EXAMPLES

CONTENTS

CONTENTS

CHAPTER 1 QUICK REFERENCE

Files by Function

Functionality	*page*	*Filename*	*Description*
Configuration			
	2-2	acfg.c	Configure a simple axis
	2-19	ampenabl.c	Configure amp enable outputs
	2-216	template.c	A simple motion control program template
Actual Velocity Calculation			
	2-6	actvel1.c	Read, calculate & print actual velocity over a 100 millisecond period
	2-8	actvel2.c	Read, calculate & print actual velocity over a 3 millisecond period
Coordinated Motion			
	2-33	ccoord.c	Perform simple circular coordinated moves
	2-100	lcoord.c	Perform simple linear coordinated moves
Counter/Timer			
	2-35	cntdown.c	Operate 8254 counter/timer in countdown mode
A/D Converter			
	2-10	adread.c	Read the A/D converter
	2-24	axisad.c	Configure DSP to read A/D converter(1 channel per axis)
Data Acquisition			
	2-159	scancap.c	Use velocity profile scan to capture positions
	2-153	scanadp.c	Trapezoidal profile motion -capture analog values at specific positions
	2-156	scanadt.c	Trapezoidal profile motion -capture analog values at specific times
	2-148	rtraj.c	Read real-time trajectory info from DSP
Debugging Routines			
	2-208	state.c	Display axis state, status, and source info
Electronic Camming			
	2-37	coil.c	Coil-winding application for 2 axes
Electronic Gearing			
	2-105	linksync.c	Electronic gearing with synchronization, using I/O sensors
	2-108	lsint.c	Link synchronization using I/O sensors in an interrupt routine
	2-200	slave.c	Command motion with linked axes

Functionality	*page*	*Filename*	*Description*
End Point Correction			
	2-26	blend.c	Adjust an axis' profile with a phantom axis
	2-29	blendpt.c	Adjust an axis' profile with a position trigger (on a phantom axis)
Exception Handling & Recovery			
	2-15	allabort.c	Generate ABORT_EVENTs using *amp_enable* bits
	2-17	allevent.c	Configure DSP to generate exception events on all axes
	2-45	eventio.c	Toggle output bit when Stop/E-Stop/Abort event occurs
	2-48	eventrec.c	Recover from simple exception event
	2-53	fbf.c	Perform feedback fault checking
	2-102	limit.c	Recover from simple limit switch event
Feed Speed Override			
	2-57	fs.c	Simple feed speed control
	2-61	holdit.c	Pause & resume motion on a path
Feedback Device Switching			
	2-21	ana2enc3.c	Switch between analog & encoder feedback
Frame Sequences			
	2-140	qmvs4.c	Download multiple frame sequences & selectively execute them
Homing Routines			
	2-64	home1.c	Homing: simple, using home input
	2-66	home2.c	Homing: two-stage, using home input
	2-68	home3.c	Homing: using combined home & index logic
	2-70	home4.c	Homing: using home & index inputs
	2-72	home5.c	Homing: using a mechanical hard stop
	2-74	home6.c	Homing: using the encoders' index pulse
	2-76	home7.c	Homing: find midpoint between positive & negative limits
Host Motion Profile Trajectory Generation			
	2-129	pvt4.c	Generate a multi-axis coordinated motion profile using frames
	2-134	pvu5.c	Use a looping frame sequence to update positions & velocities every *n* samples
I/O Configurations			
	2-41	disded.c	Disable DSP from reading/writing to dedicated output bits
	2-125	p3cfg.c	Reconfigure Dedicated I/O (axes 4 - 7) as User I/O
Interrupt Handling			
	2-13	aimint.c	Axis motion & DSP I/O monitoring with interrupt handling
	2-43	eventint.c	Generate interrupts to CPU using exception events
	2-79	inthnd.c	Single-board interrupt support
	2-82	intsimpl.c	Single-board interrupt support under DOS
	2-117	minthnd.c	Multi-board interrupt support
Jogging			
	2-89	jog.c	Use the **jog_axes(...)** function
	2-91	joglat1.c	Jog, latch positions, generate stop events, clear status on 3 axes
	2-94	joglat2.c	Jog, latch positions, generate stop events, back off a relative distance
Motion Settling			
	2-172	settle3.c	Determine when actual motion has settled

Functionality	*page*	*Filename*	*Description*
Performance Benchmarking			
	2-203	speed.c	Determine function execution time (how long)
PID Filter			
	2-127	pidchng1.c	Change PID filter parameters when error limit is exceeded
Position Latching			
	2-85	iolatch.c	Latch the actual positions of all axes using User I/O bit 22
	2-98	keylatch.c	Latch the actual positions of all axes using keyboard input
	2-211	stoplat.c	Move, latch positions, generate stop events, back off a relative distance
Redundant Feedback			
	2-145	redfed.c	Perform redundant feedback checking
SERCOS			
	2-123	oscdata1.c	Oscilloscope data acquisition with an Indramat drive
	2-150	sattr.c	Read IDN attributes
	2-162	scrstat.c	Read ring status
	2-164	sdiag.c	Read drive diagnostics & reset/enable drive
	2-170	sercpl2.c	Use position latching with a drive
	2-175	shome1.c	Perform a homing routine
	2-177	sidn.c	Decode an IDN's value based on its attributes
	2-182	sidn1.c	Read IDN value/string
	2-184	sidnl.c	Read an "IDN-List" from an IDN in a specific drive
	2-186	sidns.c	Read/write/copy a group of IDNs
	2-188	sinit1.c	Initialize ring with Indramat drive
	2-191	sinit2.c	Initialize ring with Indramat drive, Phase 2 & 3 IDNs
	2-194	sinit3.c	Initialize ring with Indramat drive, user-specified cyclic data
	2-197	sinit4.c	Initialize loop with Lutze ComCon, 16-bit I/O modules
	2-202	snfind.c	Find node addresses
Synchronized Motion			
	2-114	masync.c	Multi-axis synchronized motion
Windows NT			
	2-51	ext_int.c	Initialization -external interrupts
	2-55	fr_int.c	Initialization -frame interrupts
	2-59	getdata.c	Read a data buffer with the device driver
	2-87	iomon_in.c	Initialization for I/O-generated interrupts
	2-120	mtask.c	Multi-tasking
	2-166	sem.c	Multi-tasking & interrupts

Files by Name

Table 2-1 List of Code Files

Filename	*Description of Sample Code*	*Page*
acfg.c	Configure a simple axis	*2-2*
actvel1.c	Read, calculate & print actual velocity over a 100 millisecond period	*2-6*
actvel2.c	Read, calculate & print actual velocity over a 3 millisecond period	*2-8*
adread.c	Read the A/D converter	*2-10*
aimint.c	Axis motion & DSP I/O monitoring with interrupt handling	*2-13*
allabort.c	Generate ABORT_EVENTs using *amp_enable* bits	*2-15*
allevent.c	Configure DSP to generate exception events on all axes	*2-17*
ampenabl.c	Configure amp enable outputs	*2-19*
ana2enc3.c	Switch between analog & encoder feedback	*2-21*
axisad.c	Configure DSP to read A/D converter (1 channel per axis)	*2-24*
blend.c	Adjust an axis' profile with a phantom axis	*2-26*
blendpt.c	Adjust an axis' profile with a position trigger (on a phantom axis)	*2-29*
ccoord.c	Perform simple circular coordinated moves	*2-33*
cntdown.c	Operate 8254 counter/timer in countdown mode	*2-35*
coil.c	Coil-winding application for 2 axes	*2-37*
disded.c	Disable the DSP from reading/writing to the dedicated output bits	*2-41*
eventint.c	Generate interrupts to CPU using exception events	*2-43*
eventio.c	Toggle output bit when a Stop/E-Stop/Abort event occurs	*2-45*
eventrec.c	Recover from simple exception event	*2-48*
ext_int.c	Initialization -external interrupts under WinNT	*2-51*
fbf.c	Perform feedback fault checking	*2-53*
fr_int.c	Initialization -frame interrupts under WinNT	*2-55*
fs.c	Simple feed speed control	*2-57*
getdata.c	Read a data buffer with the device driver under WinNT	*2-59*
holdit.c	Pause & resume motion on a path	*2-61*
home1.c	Homing: simple, using home input	*2-64*
home2.c	Homing: two-stage, using home input	*2-66*
home3.c	Homing: using combined home & index logic	*2-68*
home4.c	Homing: using home & index inputs	*2-70*
home5.c	Homing: using a mechanical hard stop	*2-72*
home6.c	Homing: using the encoders' index pulse	*2-74*
home7.c	Homing: find midpoint between positive & negative limits	*2-76*
inthnd.c	Single-board interrupt support	*2-79*
intsimpl.c	Single-board interrupt support under DOS	*2-82*
iolatch.c	Latch the actual positions of all axes using User I/O bit 22	*2-85*
iomon_in.c	Initialization for I/O-generated interrupts under WinNT	*2-87*
jog.c	Use the *jog_axes(...)* function	*2-89*
joglat1.c	Jog, latch positions, generate stop events, clear status on 3 axes	*2-91*
joglat2.c	Jog, latch positions, generate stop events, back off a relative distance	*2-94*
keylatch.c	Latch the actual positions of all axes using keyboard input	*2-98*
lcoord.c	Perform simple linear coordinated moves	*2-100*
limit.c	Recover from simple limit switch event	*2-102*
linksync.c	Electronic gearing with synchronization, using I/O sensors	*2-105*
lsint.c	Link synchronization using I/O sensors in an interrupt routine	*2-108*

Additional Notes

Coordinated Motion

For applications that require *coordinated motion* between 2 or more axes that start and stop in the same position at the same time, MEI motion controllers can create a smooth path of motion.

Electronic Camming

Electronic camming links a slave axis to a master axis to simulate the motion of a mechanical cam. With electronic camming, complex, multi-turn profiles can be created, often that are not feasible with mechanical systems.

Electronic Gearing

To simulate the motion of mechanical gears and linkages electronically, MEI controllers can easily slave the position of one (or more) axes to another. Once linked, changes in either the command or actual position of the master axis are reflected in the position of the slave axis. Gearing or link ratios can be changed on-the-fly and any axis may act as a master or slave. One master axis can have multiple slave axes. Each controller can have multiple master axes.

End Point Correction

To specifically change the endpoint in DSP and DSPpro Series controllers, the motion profile of a real axis is smoothly blended with a phantom axis. The new trajectory is controlled by determining the amount of blending and calculating a trigger position for executing the phantom axis' motion profile. The DSP handles the blending without host CPU intervention.

Feed Speed Override

The feed speed override function can be used to decrease or increase speed along a path from 0% to 200%. Feed speed can be controlled from the keyboard, an analog input, or an encoder input. Use the feed speed function to pause and then resume motion (without leaving the commanded path) while motion is in progress.

Jogging

MEI controllers allow a stage or other device to be manipulated using analog data input directly from an external device (joystick, trackball, tensioner, etc.). Using data from an on-board A/D converter, the DSP converts the analog signal to a velocity command through a configurable linear and cubic term. The DSP calculates the command velocity to accomplish analog jogging without host intervention.

For trackballs and other devices that provide quadrature encoder signals instead of analog output, MEI controllers can also perform encoder-based jogging. Like analog jogging, encoder jogging offers configurable linear and cubic term for precise control.

Synchronized Motion

Point-to-point motion for a simultaneous or synchronized start can be programmed for multiple axes using trapezoidal, parabolic, or S-curve motion profiles. MEI controllers can execute multiple point-to-point moves on-board without host involvement. Multiple axes can be synchronized to each other, to a specific point in time, on an I/O point, or using a combination of these elements.

CHAPTER 2 CODE EXAMPLES

On the following pages, the contents of the sample application files are printed for your convenience. The files are listed in alphabetical order.

Quick reference tables with cross-referenced page numbers are in Chapter 1.

acfg.c

Configure a simple axis

Configuration Files

```
/* ACFG.C

:simple axis configuration.

This sample demonstrates how to configure the axes for open-loop stepper or closed-loop servo
(default) operation.

The following switches can be set from the command line:

 n -      where "n" is the number of axes to configure
 smt -    surface mount constructed board
 boot -   save changes to boot memory
 olstep - open loop stepper
 clserv - closed loop servo
 fast -   highest step pulse output range
 medium - middle step pulse output range
 slow -   lowest step pulse output range

Note:  The configuration for open/closed loop operation must occur in axis
 pairs (0 and 1, 2 and 3, etc.).

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <string.h>
# include "pcdsp.h"

struct
{  int16 axes;               /* number of axes to configure */
   int16 surface_mount;      /* surface mount or through hole controller */
   int16 boot;               /* update DSP's boot memory or dynamic memory */
} board;

struct
{  int16 step_motor;         /* servo or stepper */
   int16 closed_loop;        /* closed loop or open loop (in pairs of axes) */
   int16 step_speed;         /* fast, medium, slow, or disabled */
   int16 unipolar;           /* unipolar or bi-polar voltage output */
   int16 feedback;           /* encoder, analog, parallel, ... */
   int16 i_mode;             /* integration always or only when standing */
   int16 * coeffs;           /* PID filter parameters */
} axis;
```

```
int16
   clserv_coeffs[] = {1024, 32, 4096, 0, 0, 32767, 0, 32767, -5, 0},
   olstep_coeffs[] = {200, 20, 0, 20, 2760, 32767, 0, 32767, -3, 0},
   olstep_smt_coeffs[] = {320, 32, 0, 32, 3750, 32767, 0, 32767, -1, 0};


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;
   }
}

void config_axes(int16 n_axes, int16 * axes)
{
   int16 i;

   for (i = 0; i < n_axes; i++)
   {
       dsp_set_stepper(axes[i], axis.step_motor);
       dsp_set_closed_loop(axes[i], axis.closed_loop);

       if (axis.step_motor && (!axis.closed_loop))          /* open-loop stepper */
       {
           axis.unipolar = TRUE;
           if (board.surface_mount)
           {   axis.coeffs = olstep_smt_coeffs;
               axis.coeffs[DF_SHIFT] = ((axis.step_speed * 2) - 7);
           }
           else
           {   axis.coeffs = olstep_coeffs;
               axis.coeffs[DF_SHIFT] = ((axis.step_speed * 2) - 9);
           }
       }
       else                                                 /* closed-loop servo */
       {
           axis.step_speed = DISABLE_STEP_OUTPUT;
           axis.unipolar = FALSE;
           axis.coeffs = clserv_coeffs;
       }

       dsp_set_step_speed(axes[i], axis.step_speed);
       set_unipolar(axes[i], axis.unipolar);
       set_feedback(axes[i], FB_ENCODER);
       set_integration(axes[i], axis.i_mode);
       set_filter(axes[i], axis.coeffs);
   }
}
```

```
void config_boot_axes(int16 n_axes, int16 * axes)
{
   int16 i;

   for (i = 0; i < n_axes; i++)
   {
       dsp_set_boot_stepper(axes[i], axis.step_motor);
       dsp_set_boot_closed_loop(axes[i], axis.closed_loop);
       dsp_set_boot_step_speed(axes[i], axis.step_speed);
       set_boot_unipolar(axes[i], axis.unipolar);
       set_boot_feedback(axes[i], FB_ENCODER);
       set_boot_integration(axes[i], axis.i_mode);
       set_boot_filter(axes[i], axis.coeffs);
       mei_checksum();
   }
}

void initialize(void)
{
   int16 error_code;

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

   board.axes = dsp_axes();

   /* Default axis configuration */
   board.surface_mount = FALSE;
   board.boot = FALSE;
   axis.step_motor = FALSE;
   axis.closed_loop = TRUE;
   axis.step_speed = SLOW_STEP_OUTPUT;
   axis.unipolar = FALSE;
   axis.feedback = FB_ENCODER;
   axis.i_mode = IM_STANDING;
   axis.coeffs = clserv_coeffs;
}

void arguments(int16 argc, char * argv[])
{
   int16 i;
   char * s;

    for (i = 1; i < argc; i++)
   {  s = argv[i];
       if (atoi(s))
       {   board.axes = atoi(s);
           continue;
       }
       if (!strcmpi(s, "smt"))
       {   board.surface_mount = TRUE;
           continue;
       }
       if (!strcmpi(s, "boot"))
       {   board.boot = TRUE;
           continue;
       }
       if (!strcmpi(s, "olstep"))
       {   axis.step_motor = TRUE;
           axis.closed_loop = FALSE;
           continue;
       }
       if (!strcmpi(s, "clserv"))
       {   axis.step_motor = FALSE;
           axis.closed_loop = TRUE;
           continue;
       }
       if (!strcmpi(s, "fast"))
       {   axis.step_speed = FAST_STEP_OUTPUT;
           continue;
```

```
      }
      if (!strcmpi(s, "medium"))
      {   axis.step_speed = MEDIUM_STEP_OUTPUT;
          continue;
      }
      if (!strcmpi(s, "slow"))
      {   axis.step_speed = SLOW_STEP_OUTPUT;
          continue;
      }
      if (!strcmpi(s, "/?"))
      {   printf("\nCommand line switches: [smt] [boot] ");
          printf("[olstep (fast|medium|slow)] [clserv]\n");
          exit(0);
      }
   }
}

int16 main(int16 argc, char * argv[])
{
   int16 axes[] = {0, 1, 2, 3, 4, 5, 6, 7};

    initialize();
   arguments(argc, argv);

   config_axes(board.axes, axes);
   if (board.boot)
       config_boot_axes(board.axes, axes);

   printf("\n%d axes are configured.", board.axes);

    return 0;
}
```

actvel1.c

Read, calculate & print actual velocity over 100 msecs

Actual Velocity Calculation

```
/* ACTVEL1.C

:Read, calculate and print actual velocity over a 100 millisecond period.

A simple move is commanded and the actual velocity (counts/sec) is printed to the screen.  The
velocity is calculated using get_position() twice and dividing by the time interval using
dsp_read_dm().

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
*/

/*  Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include <dos.h>
# include "pcdsp.h"


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;
   }
}
```

actvel1.c

Read, calculate & print actual velocity over 100 msecs

```
double actual_velocity(int16 axis)
{
   double position1, position2, vel;
   int16 transfer_time1, transfer_time2;

   get_position(axis, &position1);
   transfer_time1 = dsp_read_dm(0x011E);    /* read data transfer sample time */
   delay(100);
   get_position(axis, &position2);
   transfer_time2 = dsp_read_dm(0x011E);

   vel = (((position2-position1)/(transfer_time2-transfer_time1))
            * dsp_sample_rate());

   return vel;
}


int16 main()
{
   int16 error_code;

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

   printf("Press a key to exit program \n");
   start_move(0, 100000.0, 10000.0, 100000.0);

   while(!kbhit())
   {
       printf(" %lf \r",actual_velocity(0));
   }
   getch();
   return 0;
}
```

actvel2.c

Read, calculate & print actual velocity over 3 msecs

Actual Velocity Calulation

```
/* ACTVEL2.C


:Read, calculate and print actual velocity over a 3 millisecond period.

A simple move is commanded and the actual velocity (counts/sec) is printed to the screen.  The
velocity is calculated using pcdsp_transfer_block(...), which reads the current velocity register
in the DSP's data memory.  The actual velocity is read three times and averaged.

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
*/

/*  Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include <dos.h>
# include "idsp.h"


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;
   }
}
```

actvel2.c

Read, calculate & print actual velocity over 3 msecs

```
double actual_velocity(int16 axis)
{
   P_DSP_DM current_v_addr = dspPtr->data_struct + DS_CURRENT_VEL + DS(axis); /* velocity in cts/
sample */

   DSP_DM VelA[1], VelB[1], VelC[1];
   double vel;

   pcdsp_transfer_block(dspPtr, TRUE, FALSE, current_v_addr, 1, VelA);
   pcdsp_transfer_block(dspPtr, TRUE, FALSE, current_v_addr, 1, VelB);
   pcdsp_transfer_block(dspPtr, TRUE, FALSE, current_v_addr, 1, VelC);

   vel = (( VelA[0] + VelB[0] + VelC[0])/3) * dsp_sample_rate();

   return vel;
}

int16 main()
{
   int16 error_code;

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

   printf("Press any key to exit program \n");
   start_move(0, 100000.0, 10000.0, 100000.0);

   while(!kbhit())
   {
      printf(" %lf\r", actual_velocity(0));
   }
   getch();
   return 0;
}
```

adread.c

Read the A/D converter

A/D Converter

```
/* ADREAD.C

:Read the Analog to Digital converter.

This sample code demonstrates how to read digital values from the A/D (analog to digital) con-
verter.  The values are stored into a buffer (host) and later displayed.  There are several com-
mand line switches to configure the A/D, the number of reads, and the display:

   diff -     read a differential channel
   bi -       read a bipolar channel
   dsp -      read the A/D with the DSP
   debug -    display the buffer

For example, to read a single ended A/D input with the DSP 5000 times, and redirect the output to
a file:

   adread 5000 dsp debug > results.txt

Warning!  The DSP and host CPU must NOT be configured to read the A/D converter at the same time.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include <dos.h>
# include <string.h>
# include "pcdsp.h"


# define MAX_BUFFER_SIZE     10000
# define AXIS                0
# define AD_CHANNEL          0      /* valid range 0 to 7 */

int16 atod[MAX_BUFFER_SIZE];
int16 buffer_size = 0;

int16 ad_bi = FALSE;         /* bipolar? */
int16 ad_diff = FALSE;       /* differential? */
int16 ad_dsp = FALSE;        /* read with the DSP? */
int16 debug = FALSE;         /* display the buffer? */
```

```
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;
   }
}

void calc_values(int16 num, int16 * value)
{
   double i, avg, max, min, total = 0.0;

   for (i = 0; i < num; i++)
       total += value[i];

   avg = total / num;
   max = avg;
   min = avg;

   for (i = 0; i < num; i++)
   {
       if (value[i] > max)
       {   max = value[i];
           if (debug)
               printf("Max: ");
       }

       if (value[i] < min)
       {   min = value[i];
           if (debug)
               printf("Min: ");
       }
       if (debug)
           printf("%4d\n", value[i]);
   }
   printf("\nDiff: %d Bi: %d DSP: %d", ad_diff, ad_bi, ad_dsp);
   printf("\nAvg: %6.2lf Max: %6.2lf Min: %6.2lf Range: %6.2lf", avg, max, min, max - min);

}

void arguments(int16 argc, char * argv[])
{
   int16 i;
   char * s;

    for (i = 1; i < argc; i++)
   {   s = argv[i];
       if (atoi(s))
       {   buffer_size = atoi(s);
           continue;
       }
       if (!strcmpi(s, "diff"))
       {   ad_diff = TRUE;
           continue;
       }
       if (!strcmpi(s, "bi"))
       {   ad_bi = TRUE;
           continue;
       }
       if (!strcmpi(s, "dsp"))
```

```
        {   ad_dsp = TRUE;
            continue;
        }
        if (!strcmpi(s, "debug"))
        {   debug = TRUE;
            continue;
        }
        if (!strcmpi(s, "?"))
        {   printf("\nUsage: [diff] [bi] [dsp] [debug]");
            exit(0);
        }
    }
    if (buffer_size > MAX_BUFFER_SIZE)
        buffer_size = MAX_BUFFER_SIZE;
    if (buffer_size < 1)
        buffer_size = 1;
}

int16 main(int16 argc, char * argv[])
{
    int16 error_code, i, dsp_time;

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

    printf("\nCollecting %d A/D values\n", buffer_size);
    set_axis_analog(AXIS, ad_dsp);

    if (ad_dsp)
    {   set_feedback(AXIS, FB_ANALOG);
        set_analog_channel(AXIS, AD_CHANNEL, ad_diff, ad_bi);
    }
    else
        init_analog(AD_CHANNEL, ad_diff, ad_bi);

    for (i = 0; i < buffer_size; i++)
    {
        if (ad_dsp)
            read_axis_analog(AXIS, &atod[i]);
        else
            start_analog(AD_CHANNEL);

        delay(1);                   /* wait for the A/D conversion (at least 15 microsec) */

        if (!ad_dsp)
            read_analog(&atod[i]);
    }
    calc_values(buffer_size, atod);

     return 0;
}
```

aimint.c

Axis motion & DSP I/O monitoring with interrupts

Interrupt Handling

```
/* AIMINT1.C

:Axis motion and DSP I/O monitoring with interrupt handling.

Sample program to demonstrate simple interrupt handling.  Motion is commanded with start_move(...)
and the DSP is configured to monitor the User I/O. When the dsp_irq_frame(...) is executed by the
DSP or any of the specified I/O bits change the DSP generates an interrupt to the PC.

The interrupt routine and other functions used by this sample are in the file INTHND.C.

The interrupt handler updates the variable io_interrupt and axis_interrupt. After the interrupt is
handled, then this sample examines io_interrupt and axis_interrupt[] to determine the cause.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include <dos.h>
# include "pcdsp.h"

extern int16 dsp_irq_frame(int16 axis);
extern void install_dsp_handler(int16 into);
extern int16 axis_interrupts[PCDSP_MAX_AXES];
extern int16 io_interrupts;
extern int16 interrupt_number = 5;                /* IRQ 5 */


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;
   }
}
```

```
int16 display(int16 axis, int16 port, int16 expected_value)
{
   int16 done = 0, key, value, status;
   double cmd;

   while (!done)
   {   get_io(port, &value);
       get_command(axis, &cmd);
       printf("Axis:%d Cmd:%8.0lf Port:%d Val:%d\r", axis, cmd, port, value);

       if (io_interrupts)                   /* updated by interrupt routine */
       {   set_io(port, expected_value);    /* set bit(s) to expected state */
           clear_io_mon();                  /* reset the DSP's I/O monitoring */
           io_interrupts = 0;               /* reset interrupt counter */
           io_mon(port, &status);
           printf("\nInterrupt--Port: %d Status: 0x%x\n\n", port, status);
       }

       if (axis_interrupts[axis])           /* updated by interrupt routine */
       {   printf("\nInterrupt--Cmd: %8.0lf\n\n", cmd);
           axis_interrupts[axis] = 0;       /* reset interrupt counter */
       }

       if (kbhit())
       {   if (getch() == 0x1B)
              done = TRUE;
           else
              set_bit(0);
       }
   }
   return 0;
}

int16 main()
{
   int16 error_code, port, mask, expected_value, axis = 0;

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

   error(dsp_reset());

   init_io(0, IO_OUTPUT);
   init_io(2, IO_OUTPUT);                   /* PC interrupt bit is located on port 2 */

   install_dsp_handler(interrupt_number);   /* use IRQ 5 */
   reset_bit(23);                           /* enable interrupts */

   set_io(0, 0x0);
   mask = 0x1;                              /* monitor bit 0, on port 0 */
   port = 0;
   expected_value = 0x0;
   set_io_mon_mask(port, mask, expected_value);

   printf("\nPress any key to set bit high, <esc> to quit");
   printf("\nMonitoring = port:%d mask:0x%x\n\n", port, mask);
   start_move(axis, 1000.0, 500.0, 1000.0);
   dsp_irq_frame(axis);

   display(axis, port, expected_value);
   set_bit(23);                             /* disable interrupts */

   return 0;
}
```

allabort.c

Generate ABORT_EVENTs using *amp_enable* bits

Exception Handling & Recovery

```
/* ALLABORT.C

:Generate ABORT_EVENTs based on the amp_enable bits.

This sample creates a sequence of frames on a phantom axis.  The sequence of frames generate
ABORT_EVENTs on axes (0-2) when any of the amp_enable's on axes 0-2 are low.  This is very useful
during coordinated motion.

The phantom axis is created by downloading 4axis.abs to a 3 axis board.

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
*/

/*  Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


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

# define  PHANTOM 3
# define  AXES    4


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;
   }
}

void disable_hardware_limits(int16 axis)
{
   set_positive_limit(axis, NO_EVENT);
   set_negative_limit(axis, NO_EVENT);
   set_home(axis, NO_EVENT);
   set_amp_fault(axis, NO_EVENT);
}
```

allabort.c

Generate ABORT_EVENTs using amp_enable bits

```
void disable_software_limits(int16 axis)
{
   int16 action;
   double position;

   get_positive_sw_limit(axis, &position, &action);
   set_positive_sw_limit(axis, position, NO_EVENT);
   get_negative_sw_limit(axis, &position, &action);
   set_negative_sw_limit(axis, position, NO_EVENT);
   get_error_limit(axis, &position, &action);
   set_error_limit(axis, position, NO_EVENT);
}

int16 main()
{
   int16 error_code, axis;

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

   disable_hardware_limits(PHANTOM);        /* prevent unintended events */
   disable_software_limits(PHANTOM);

   for (axis = 0; axis < AXES; axis++)
   {
       set_amp_enable_level(axis, TRUE);
       while (controller_run(axis))
           ;
       enable_amplifier(axis);
   }

/* The amp enable output bits for axes 0, 1 and 2 (bits 0, 2 and 4) are on
   user I/O port 8.  The value for the amp enables mask = 0x1 + 0x4 + 0x10.
   When the state parameter is FALSE, the next frame is triggered if any
   masked bit is low (0 volts).
*/
   dsp_io_trigger_mask(PHANTOM, 8, 0x15, FALSE);

   dsp_axis_command(PHANTOM, 0, ABORT_EVENT);
   dsp_axis_command(PHANTOM, 1, ABORT_EVENT);
   dsp_axis_command(PHANTOM, 2, ABORT_EVENT);

    return 0;
}
```

allevent.c

Configure DSP to generate exception events

(on all axes)

Exception Handling & Recovery

```
/* ALLEVENT.C

:Configure the DSP to generate exception events on all axes.

The Stop, E-Stop, and Abort Events are executed by the DSP based on an axis' limit switches, home
switch, software limits, error limits, etc.  Internally, the DSP executes some special frames to
complete the exception event.

When a Stop Event or E-Stop Event is generated, the DSP executes a Stop or E-Stop frame to decel-
erate the axis.  Then the DSP executes the Stopped frame to set the command velocity to zero.  An
Abort Event puts the axis in idle mode (no PID update) and executes the Stopped frame.  See the "C
Programming" manual for more information about Exception Events.

By modifying an axis' Stopped exception frame, an exception event can be sent to another axis by
the DSP after a Stop, E-Stop, or Abort Event executes.

This code configures the Stopped frame for each axis to generate an Abort Event on the next
sequential axis.  Thus, if an Abort Event is generated on any axis, then Abort Events will be gen-
erated on all axes.  Also, if a Stop or E-Stop Event is generated on an axis, then Abort Events
will be generated on all axes (after the original axis decelerates to a stop).

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.
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"
# include "idsp.h"


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;
```

```
   }
}

int16 config_stopped_frame(PFRAME f, int16 address, int16 ormask, int16 andmask)
{
   int16 update = f->f.trig_update & 0xFF;          /* get trigger/update field */
   int16 * i = (int16*) &(f->f.position);           /* pointer to position field */

   if (!(update & FUPD_POSITION))
   {
       f->f.output = OUTPUT_POSITION ;              /* set output to use position field */
       i[0] = address;
       i[1] = ormask;
       i[2] = andmask;
       f->f.trig_update |= FUPD_OUTPUT;             /* update register from position field */
       pcdsp_write_ef(dspPtr, f);                   /* download the exception frame */
       return DSP_OK ;
   }
   return DSP_RESOURCE_IN_USE;
}

int16 set_stopped_frame(int16 axis, int16 destaxis, int16 event)
{
   FRAME f;

   event |= (ID_AXIS_COMMAND << 4);

   if (pcdsp_sick(dspPtr, axis) || pcdsp_sick(dspPtr, destaxis))
       return dsp_error;

   if (pcdsp_read_ef(dspPtr, &f, axis, EF_STOPPED))
       return dsp_error;

   if (config_stopped_frame(&f, dspPtr->pc_event + destaxis, event, event))
       return dsp_error;
   else
       return pcdsp_write_ef(dspPtr, &f);
}

int16 main()
{
   int16 error_code, axis, axes, dest_axis;

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

   axes = dsp_axes();

   for (axis = 0; axis < axes; axis++)
   {
       dest_axis = (axis + 1) % axes;
       set_stopped_frame(axis, dest_axis, ABORT_EVENT);
       printf("\n%d %d", axis, dest_axis);
   }

    return 0;
}
```

ampenabl.c

Configure amp enable outputs

Configuration files

```
/* AMPENABL.C

:Demonstrates how to configure the dedicated amp enable outputs.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/

# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"


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;
   }
}

int16 get_info(void)
{
   char  buffer[80];
   int16   generic;

   gets(buffer);
   sscanf(buffer,"%d",&generic);
   return generic;
}
```

ampenabl.c

Configure amp enable outputs

```
int16 enable_axis(int16 axis)
{
   int16      level;

   printf("\nEnter run time amp enable level for axis %d (1 = HIGH, 0 = LOW): ", axis);
   level = get_info();
   set_amp_enable_level(axis, level);
   printf("\nSet amp enable to disabled state for boot up? (y/n) ");
   if(getch() == 'y')
   {
      set_boot_amp_enable_level(axis, level);
      // NOTE: the call to set_boot_amp_enable() is not really needed since
      // set_boot_amp_enable_level() calls set_boot_amp_enable() with !level.
      set_boot_amp_enable(axis, !level);
      mei_checksum();
   }
   enable_amplifier(axis);

   return dsp_error;
}

int16 main()
{
   int16      i, level, bootlevel, state, bootstate;
   int16 error_code;

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

   for(i = 0; i < dsp_axes(); i++)
      if(enable_axis(i))
         printf("\nCould configure amp enable for axis %d.\n", i);
      else
      {
         printf("\nAmp enable configured for axis %d.\n", i);
         get_amp_enable_level(i, &level);
         printf("Runtime amp enable level is %s\n", level ? "HIGH" : "LOW");
         get_amp_enable(i, &state);
         printf("Runtime amp enable state is %s\n", state ? "HIGH" : "LOW");
         get_boot_amp_enable_level(i, &bootlevel);
         printf("Boot amp enable level is %s\n", bootlevel ? "HIGH" : "LOW");
         get_boot_amp_enable(i, &bootstate);
         printf("Boot amp enable state is %s\n", bootstate ? "HIGH" : "LOW");
      }

   return 0;
}
```

ana2enc3.c

Switch between analog & encoder feedback

Feedback Device Switching

```
/* ANA2ENC3.C

:Switching between analog and encoder feedback.

This sample code configures the first axis, ENCODER_AXIS for encoder feedback and the second axis,
ANALOG_AXIS for analog feedback.  Based on keyboard input the +/-10 volt analog output channels
are switched between the ENCODER_AXIS and the ANALOG_AXIS.

The encoder input is used for position control and the analog feedback is typically used for force
control.  By switching the analog output channels we are really switching feedback devices, each
with their own PID loop.  The motor is controlled by a single analog control voltage output, the
DAC_CHANNEL.  The ANALOG_AXIS' analog output channel is not used.

Note:  The default analog output configuration is axis = dac channel.

Since the analog output for the ANALOG_AXIS is not used, it does not require the digital to analog
circuitry.  For example, a single axis controller can control one motor with one channel of
encoder feedback and one channel of analog feedback.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$
   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"

# define AXES    2

# define ENCODER_AXIS       0       /* encoder feedback axis */
# define ANALOG_AXIS        1       /* analog feedback axis */

# define DAC_CHANNEL        0       /* DAC channel to control motor */


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;
    }
}

void display_dacs(int16 axes)
{
    int16 axis, value;

    for (axis = 0; axis < axes; axis++)
    {
        get_dac_output(axis, &value);
        printf("Axis%d Dac%d  ", axis, value);
    }
    printf("\n");
}

void display_positions(int16 axes)
{
    int16 axis;
    double cmd, act;

    printf("\r");
    for (axis = 0; axis < axes; axis++)
    {
        get_command(axis, &cmd);
        get_position(axis, &act);
        printf("Axis%d Cmd%6.0lf Act%6.0lf  ", axis, cmd, act);
    }
}

void set_control_axis(int16 control_axis, int16 other_axis)
{
    int16 dac_a, dac_b;

    /* Read the configured dac channel for each axis. */
    get_dac_channel(control_axis, &dac_a);
    get_dac_channel(other_axis, &dac_b);

    if (dac_a != DAC_CHANNEL)               /* switch the dac channels? */
    {
    /* Set the command position equal to the actual position to prevent the
        motor from "snapping" to position.
    */
        controller_run(control_axis);
        controller_run(other_axis);

        display_dacs(AXES);

    /* Switch the dac channel configuration.  Be sure to always change the
        control axis first.  By switching the dac channels we are really
        switching feedback devices, each with their own PID loop.
    */
        set_dac_channel(control_axis, dac_b);
        set_dac_channel(other_axis, dac_a);
    }
}
```

```
int16 main()
{
   int16 error_code, done, key;

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

   set_feedback(ANALOG_AXIS, FB_ANALOG);
   set_analog_channel(ANALOG_AXIS, 0, FALSE, FALSE);

   printf("\na=analog feedback, e=encoder feedback, <esc> to quit.\n");
   for (done = 0; !done; )
   {
       display_positions(AXES);

       if (kbhit())/* key pressed? */
       {   key = getch();
           switch (key)
           {
                 case 'e':
                      printf("\n\nEncoder feedback control. \n");
                      set_control_axis(ENCODER_AXIS, ANALOG_AXIS);
                     break;

                 case 'a':
                      printf("\n\nAnalog feedback control.  \n");
                      set_control_axis(ANALOG_AXIS, ENCODER_AXIS);
                     break;

              case 0x1B:                    /* <ESC> */
                     done = TRUE;
                      break;
           }
       }
   }

    return 0;
}
```

Switch between analog & encoder feedback

axisad.c

Configure DSP to read A/D (1 channel per axis)

A/D Converter

```
/* AXISAD.C

:Configure the DSP to read A/D (one channel per axis)

This code configures the DSP to read 1 analog input channel per axis. The analog input channels
are configured for unipolar voltage, single-ended inputs.  The A/D value is displayed to the
screen.

Remember, the CPU and the DSP cannot directly read from the A/D converter at the same time.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"


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;
   }
}
```

```
void display(int16 axes)
{
   int16 axis, value;

   while (!kbhit())
   {   for (axis = 0; axis < axes; axis++)
       {   read_axis_analog(axis, &value);
           printf("%1d:%4d ", axis, value);
       }
       printf("\r");
   }
   getch();

}

int16 main()
{
   int16 error_code, axis, axes;

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

   error(dsp_reset());              /* reset the hardware */

   axes = dsp_axes();

   for (axis = 0; axis < axes; axis++)
   {   set_analog_channel(axis, axis, FALSE, FALSE);
       set_axis_analog(axis, TRUE);         /* enable the DSP to read the A/D */
   }

   display(axes);

    return 0;
}
```

blend.c

Adjust an axis' profile with a phantom axis

Endpoint Correction

```
/* BLEND.C

:Sample to adjust an axis' profile with a phantom axis.

Axis 0 is the command axis, which is linked to Axis 3's command position.  While axis 0 performs
a long move, axis 3 is advanced based on keyboard input.  All that is required from axis 3 is its
command position.  The simulated axis is configured by downloading 4axis.abs (4 axis firmware) to
a 3 axis card.

Note: Disable all hardware and software limits for the phantom axis.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include <dos.h>
# include "pcdsp.h"

# define  READY      0
# define  GO         1
# define  MOVE_FWD   2
# define  MOVE_BACK  3

# define  PHANTOM    3       /* simulated axis */
# define  SLAVE      0

int16  state;                /* one state variable for all axes. */

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;
   }
```

```
}

int16 CheckState(void)
{
    double act, offset;

    get_position(SLAVE, &act);
    get_command(PHANTOM, &offset);
    printf("Axis: %8.0lf Offset: %8.0lf\r", act, offset);

   switch (state)
   {
       case GO:
           start_move(SLAVE, 20000.0, 1000.0, 10000.0);
           start_move(SLAVE, 0.0, 1000.0, 10000.0);
           state = READY;
           break;

         case MOVE_FWD:
             start_r_move(PHANTOM, 500.0, 1000.0, 10000.0);
             state = READY;
             break;

         case MOVE_BACK:
             start_r_move(PHANTOM, -500.0, 1000.0, 10000.0);
             state = READY;
             break;

        case READY:
           break;

    }
    return(0);
}

int16 main()
{
   int16 error_code, axis, done, key;

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

    mei_link(PHANTOM, SLAVE, 1.0, LINK_COMMAND);
    printf("\ng=go, f=fwd, b=back, esc=quit\n");

   for (done = 0; !done; )
   {
       CheckState();

       if (kbhit())                 /* key pressed? */
       {   key = getch() ;

           switch (key)
           {
                 case 'g':          /* Go axis 0 */
                   state = GO;
                   break;

                 case 'f':          /* adjust the SLAVE forward */
                   state = MOVE_FWD;
                   break;

                 case 'b':          /* adjust the SLAVE backward */
                   state = MOVE_BACK;
                   break;

              case 0x1B:/* <ESC> */
                  done = TRUE;
                  break;
           }
```

```
        }
    }
    return 0;
}

                case 'b':               /* adjust the SLAVE backward */
                    state = MOVE_BACK;
                    break;

                case 0x1B:              /* <ESC> */
                    done = TRUE;
                    break;
            }
        }
    }
    return 0;
}
```

blendpt.c

Adjust an axis profile with a position trigger

(on a phantom axis)

Endpoint Correction

```
/* BLENDPT.C

:Sample to adjust an axis' profile with a position trigger on a phantom axis.

During the execution of a motion profile on the real axis, the profile and final position can be
adjusted by using a phantom axis.  The real axis motion profile is blended by positionally linking
a real axis to a phantom axis. Then motion is commanded on both axes.  The DSP handles the linking
(blending)
 without host CPU intervention.

Smooth profile blending is achieved by coordinating the motion profile of the real axis with the
phantom axis.  This is controlled by determining the amount of blending and calculating a trigger
position for the execution of the phantom axis' motion profile.

The trigger position is calculated based on the distance offset:

 1)  Small increase in distance
 2)  Large increase in distance
 3)  Small decrease in distance
 4)  Medium decrease in distance
 5)  Large decrease in distance

This method of profile blending works best for small adjustments to the real axis' profile.  Since
the same acceleration and velocity parameters are used for both the real and phantom axes, the
blended velocity will not exceed the original real axis command velocity.

A phantom axis can be created by downloading (n+1) axis firmware to an (n) axis board.  All that
is required from the phantom axis is its command position. Be sure to disable all hardware and
software limits for the phantom axis.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include <dos.h>
# include <math.h>
# include "idsp.h"


# define FINAL    (30000.0)
# define VEL      10000.0
# define ACCEL    10000.0
```

```
# define OFFSET  (8000.0)

# define PHANTOM     2       /* simulated axis */
# define REAL        0

struct
{  double start;
   double final;
   double vel;
   double accel;
} params[PCDSP_MAX_AXES];


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;
   }
}

void disable_hardware_limits(int16 axis)
{
   set_positive_limit(axis, NO_EVENT);
   set_negative_limit(axis, NO_EVENT);
   set_home(axis, NO_EVENT);
   set_amp_fault(axis, NO_EVENT);
}

void disable_software_limits(int16 axis)
{
   int16 action;
   double position;

   get_positive_sw_limit(axis, &position, &action);
   set_positive_sw_limit(axis, position, NO_EVENT);
   get_negative_sw_limit(axis, &position, &action);
   set_negative_sw_limit(axis, position, NO_EVENT);
   get_error_limit(axis, &position, &action);
   set_error_limit(axis, position, NO_EVENT);
}

void display(int16 axis, int16 phantom, int16 n_values)
{
   int16 i;
   double cmda, cmdb, vela, velb;

   for (i = 0; i < n_values; i++)
   {
       get_command(axis, &cmda);
       get_command(phantom, &cmdb);
       get_velocity(axis, &vela);
       get_velocity(phantom, &velb);
       printf("\nC:%8.0lf V:%8.0lf C:%8.0lf Vel:%8.0lf", cmda, vela, cmdb, velb);
   }
}
```

```
int16 adjust_profile(int16 axis, int16 phantom, double offset)
{
   int16 mdir, odir, sense;
   double dist, decel_dist, decel_time, trigger;

   if (params[axis].final > params[axis].start)    /* travel direction */
   {   mdir = 1;
       sense = POSITIVE_SENSE;
   }
   else
   {   mdir = -1;
       sense = NEGATIVE_SENSE;
   }

   if (offset > 0.0)         /* find the offset direction */
       odir = 1;
   else
       odir = -1;

   decel_time = params[axis].vel / params[axis].accel;
   decel_dist = params[axis].accel * decel_time * decel_time * .5;
   dist = mdir * (params[axis].final - params[axis].start);

   if (mdir == odir)         /* Increase the move distance */
   {
       if ((odir * offset) >= (2.0 * decel_dist))
           trigger = params[axis].final - (mdir * decel_dist);
       else
           trigger = params[axis].final - (offset / 2.0);
   }
   if (mdir != odir)         /* Decrease the move distance */
   {
       if ((odir * offset) >= (dist - decel_dist))
           trigger = params[axis].final - (mdir * dist) + (mdir * decel_dist);

       if ((odir * offset) < (dist - decel_dist))
           trigger = params[axis].final - (mdir * decel_dist) + offset;

       if ((odir * offset) < (2.0 * decel_dist))
       {
           trigger = params[axis].final - (mdir * params[axis].vel *
               (sqrt(fabs(offset) / params[axis].accel))) -
               (mdir * decel_dist);
       }
   }

   printf("\nDist:%8.0lf Offset:%8.0lf\n", dist, offset);
   printf("Decel Dist: %lf Trig:%lf\n", decel_dist, trigger);
   dsp_position_trigger(phantom, axis, trigger, sense, FALSE, NEW_FRAME);

   start_r_move(phantom, offset, params[axis].vel, params[axis].accel);

   return 0;
}
```

blendpt.c

Adjust an axis profile with a position trigger

```
int16 main()
{
   int16 error_code, axis, done, key;

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

   disable_hardware_limits(PHANTOM);
   disable_software_limits(PHANTOM);
   controller_run(PHANTOM);

    mei_link(PHANTOM, REAL, 1.0, LINK_COMMAND);

   params[REAL].start = 0.0;
    params[REAL].final = FINAL;
    params[REAL].vel = VEL;
    params[REAL].accel = ACCEL;

    start_move(REAL, params[REAL].final, params[REAL].vel, params[REAL].accel);
   display(REAL, PHANTOM, 50);
    adjust_profile(REAL, PHANTOM, OFFSET);
   while (!motion_done(REAL) || !motion_done(PHANTOM))
       display(REAL, PHANTOM, 5);

   return 0;
}
```

ccoord.c

Perform simple circular coordinated moves

Coordinated Motion

```
/* CCOORD.C

:Some simple circular coordinated moves

This code initializes the mapped axes, the vector velocity, and the vector acceleration.  Then a
single circular coordinated move is performed followed by a few interpolated circular coordinated
motion.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"


# define  MAX_AXES   2

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;
   }
}

int16 display(void)
{
   double x, y;

    while (!kbhit())
    {   get_command(0, &x);
        get_command(1, &y);
        printf("X: %12.4lf Y: %12.4lf\r", x, y);
    }
```

ccoord.c

Perform simple circular coordinated moves

```
    getch();

    return 0;
}

int16 main()
{
   int16 error_code, axes[MAX_AXES] = {0, 1};

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

    error(dsp_reset());             /* reset the hardware */

   error(map_axes(MAX_AXES, axes)); /* initialize the coordinated axes */
   set_move_speed(200.0);           /* set the vector velocity */
   set_move_accel(8000.0);          /* set the vector acceleration */
   set_arc_division(10.0);          /* arc segment every 10 degrees */

   arc_2(0.0, 100.0, 90.0);         /* perform a single coordinated move */
    display();

   /*  Perform an interpolated motion through several points */
   start_point_list();
    arc_2(100.0, 0.0, 45.0);
    arc_2(200.0, 100.0, -90.0);

    end_point_list();

    start_motion();

    display();

    return 0;
}
```

cntdown.c

Operate 8254 counter/timer in countdown mode

Counter/Timer

```
/* CNTDOWN.C

:Demonstrates the 8254 counter/timer in countdown mode.

This code toggles user I/O bit #0 when 'd' is pressed.  The hardware user I/O bit #0 should be
connected to Clock 0 of Channel 0.  Gate 0 should be connected with a pull-up resistor (1K) to +5
volts.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"


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;
   }
}

int16 main()
{
   int16 error_code, key, done;
    unsigned16 value;

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

    init_io(0, IO_OUTPUT);
    set_io(0, 0);
    init_timer(0, 0);
```

```
    set_timer(0, 2000);             /* load the counter/timer with a start value */

    printf("\nd=decrement timer, esc=exit\n");

  for (done = 0; !done; )
  {
       get_timer(0, &value);
       printf("Value: %7u\r", value);

      if (kbhit())                  /* key pressed? */
      {   key = getch();

          switch (key)
          {
             case 'd':              /* decrement timer */
                 set_bit(0);
                    reset_bit(0);
                    break;

             case 0x1B:             /* <ESC> */
                 done = TRUE;
                 break;
          }
      }
  }

    return 0;
}
```

coil.c

Coil-winding application for 2 axes

Electronic Camming

```
/* COIL.C

:Sample coil winding application for 2 axes.

Axis 0 is the master and axis 1 is the cam.  As the master rotates the cam axis moves back and
forth to place the wire evenly (similar to a fishing reel) on the master drum.

For this sample an acceleration and deceleration of the cammed axis is calculated to require
exactly one sample.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include <dos.h>
# include "pcdsp.h"

# define MASTER              0
# define CAM                 1

# define CAM_DIST            1000.0

# define COUNTS_PER_REV      4000.0
# define REVS_PER_LAYER      5.0
# define LAYERS              2.0
# define MASTER_VEL          12500.0
# define MASTER_ACCEL        100000.0


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;
```

```
   }
}

int16 display(void)
{
   int16 m_flag = 0, c_flag = 0;
   double master, cam, acam;

    while (!kbhit())
    {   get_command(MASTER, &master);
        get_command(CAM, &cam);
        get_position(CAM, &acam);
        if((master > 40000.0) & (m_flag == 0))
        {
            printf("\nCam when master is at 40000: %12.2lf\n", cam);
            m_flag = 1;
        }
        if((master > 1000) & (cam == 0) & (c_flag ==0))
        {
            printf("\nCam at zero master at: %12.2lf\n", master);
            c_flag = 1;
        }
        printf("Master: %12.2lf Cam: %12.2lf Cam Actual: %12.2lf\r", master, cam, acam);
    }
    getch();

    return 0;
}

int16 main()
{
    int16 error_code, i, dir;
    double  cam_time, cam_vel, cam_accel, sample_rate, time_ratio, master_dist;
    double  one_master_sample_CAM_time;

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

    sample_rate = (double)dsp_sample_rate();


/*****************************************************************************
Calculate the distance for the motion of master axis.  Since the change in
   position of the master axis is being substituted for the time of the cam
   axis we are really calculating the time for the cam to wind one layer.
*****************************************************************************/

   cam_time = COUNTS_PER_REV * REVS_PER_LAYER / sample_rate;

/*****************************************************************************
Calculate the time factor ratio between the cam and the master axis.

   one count of "real" time = (1 / sample_rate) * (1 sample) = 1 sec
   one count of "cam" time = (1 / sample_rate) * (1 cam_sample)
   1 cam_sample = 1 master encoder count
   one count of "cam" time = (1 / sample_rate) * (1 master count)

Taking the ratio between "cam" and "real" times:

   time_ratio = (1 master count) / (1 sample)
   time_ratio = ((1 master count) / (1 sample)) * ((1 sec) / (1 sec))
   time_ratio = (1 master count / sec) / (1 sample / sec)
   time_ratio = (master velocity) / (sample_rate)
*****************************************************************************/

   time_ratio = MASTER_VEL / sample_rate;

/*****************************************************************************
Convert the time interval for one sample on the master axis to the cam time:
```

```
   one_master_sample_CAM_time =
       one_master_sample_REAL_time * MASTER_VEL / sample_rate
   one_master_sample_CAM_time = one_master_sample_REAL_time * time_ratio
   one_master_sample_REAL_time = 1 / sample_rate
   one_master_sample_CAM_time = (1 / sample_rate) * time_ratio
*****************************************************************************/

   one_master_sample_CAM_time = time_ratio / sample_rate;

/*****************************************************************************
Now the velocity can be calculated by vel = dist / time.  For the cam velocity:

   cam_vel = CAM_DIST / cam_slew_time

   In order for the cam axis and the master axis to reach their respective
   endpoints at the same time, the amount of time passed in 4 samples on the
   master axis must be subtracted from the cam_time.  This is because the
   previous calculation for the cam_time is the amount of time allotted for a
   move on the cam axis.  Two of the frames subtracted are for the acceleration
   and deceleration of the cam axis.  The other two frames are dwell frames
   between the moves.

   cam_slew_time = cam_time - (4 * one_master_sample_CAM_time)
   cam_vel = CAM_DIST / (cam_time - (4 * one_master_sample_CAM_time))
*****************************************************************************/

   cam_vel = CAM_DIST / (cam_time - (4.0 * one_master_sample_CAM_time));

/*****************************************************************************
The acceleration can be calculated by accel = vel / time.  In the case of the
   cam axis:

   cam_accel = cam_vel / cam_accel_time

   The cam_accel_time is the time passed on the cam axis during on sample of
   the master axis, which is calculated above as one_master_sample_CAM_time.

   cam_accel_time = one_master_sample_CAM_time
   cam_accel = cam_vel / one_master_sample_CAM_time
*****************************************************************************/

   cam_accel = cam_vel / one_master_sample_CAM_time;

   printf("READY? (press any key to start)\n");
   getch();                                        /* hit a key to start */

   set_cam(MASTER, CAM, TRUE, CAM_ACTUAL);         /* enable the cam axis */

   dir = 1;
   for (i = 0; i < LAYERS; i++)
   {   start_r_move(CAM, dir * CAM_DIST, cam_vel, cam_accel);
       dir = -dir;
   }

   master_dist = cam_time * LAYERS * sample_rate;
   start_r_move(MASTER, 2*master_dist, MASTER_VEL, MASTER_ACCEL/100);
   display();

   return 0;
}


/*****************************************************************************
For camming applications, time on the cam axis is no longer "real" time.  Time
   is instead based on the change in position of the master axis.  Each
   position count of the master axis is translated into a sample count on the
   slave axis.  For example: If the master is moving at a velocity of 10 counts
   per second, then in one second of "real" time, the time passed on the cammed
   axis is equal to the amount of time it takes for 10 samples to execute.
```

```
   This "cam" time can be calculated, but it is much easier to think in terms
   of the time_ratio calculated in the above sample program.  Using this
   time_ratio, the following chart outlines the basic transformation equations
   from "real" time to "cam" time:

      "REAL" Time               "CAM" Time
-------------------------------------------------------------------------------
      MASTER_DISTANCE      |  MASTER_DISTANCE
      MASTER_VELOCITY      |  MASTER_VELOCITY
      MASTER_ACCELERATION  |  MASTER_ACCELERATION
      MASTER_JERK          |  MASTER_JERK
-------------------------------------------------------------------------------
      CAM_DISTANCE         |  CAM_DISTANCE
      CAM_VELOCITY         |  CAM_VELOCITY / time_ratio
      CAM_ACCELERATION     |  CAM_ACCELERATION / (time_ratio * time_ratio)
      CAM_JERK             |  CAM_JERK / (time_ratio * time_ratio * time_ratio)
-------------------------------------------------------------------------------
*******************************************************************************/
```

disded.c

Disable DSP from R/W to dedicated output bits

I/O Configuration

```
/* DISDED.C

:Disable the DSP from reading/writing to the dedicated output bits.

Each axis has 2 dedicated output bits (in-position and amplifier enable).  The dedicated output
bits for axes 0-3 are located on I/O port 8 and the dedicated output bits for axes 4-7 are located
on I/O port 5.

Normally, the DSP reads each dedicated output port (once per axis), masks the 8 bits, and writes
to each dedicated output port.  When disabling the dedicated output bits, be sure to disable the
axes in groups of four (0-3 and/or 4-7).

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include "pcdsp.h"
# include "idsp.h"


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;
   }
}
```

disded.c

Disable DSP from R/W to dedicated output bits

```
int16 main()
{
   int16 error_code, axis, axes, ded_addr;

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

   axes = dsp_axes();

   for (axis = 0; axis < axes; axis++)
   {
       ded_addr = dspPtr->e_data + ED_STATUS_PORT + DS(axis);
       dsp_write_dm(ded_addr, 0x0);
   }

    return 0;
}
```

eventint.c

Generate interrupts to CPU using exception events

Interrupt Handling

```
/* EVENTINT.C

:Generates interrupts to the CPU based on exception events.

Sample program to demonstrate simple interrupt handling.  This code generates a STOP_EVENT on the
DSP which generates an interrupt to the CPU.

The interrupt routine and other functions used by this sample are in the file INTHND.C.

The interrupt handler updates the variable io_interrupt and axis_interrupt. After the interrupt is
handled, then this sample examines io_interrupt and axis_interrupt[] to determine the cause.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include <dos.h>
# include "pcdsp.h"

extern int16 dsp_irq_frame(int16 axis);
extern void install_dsp_handler(int16 into);
extern int16 axis_interrupts[PCDSP_MAX_AXES];
extern int16 io_interrupts;
extern int16 interrupt_number = 5;                /* IRQ 5 */


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;
   }
}
```

eventint.c

```
int16 main()
{
   int16 error_code;

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

   error(dsp_reset());

   init_io(2, IO_OUTPUT);           /* PC interrupt bit is located on port 2 */
   set_io(2, 0x0);

   install_dsp_handler(interrupt_number);          /* use IRQ 5 */

   interrupt_on_event(0, TRUE);

   printf("Int:%d Gen:%d\n", axis_interrupts[0], io_interrupts);
   getch();

   set_stop(0);

   while (!kbhit())
       printf("Int:%d Gen:%d\r", axis_interrupts[0], io_interrupts);
   getch();

   return 0;
}
_
```

Generate interrupts to CPU using exception events

eventio.c

Toggle output bit when Stop/E-Stop/Abort event occurs

Exception Handling & Recovery

```
/* EVENTIO.C

:Toggles an output bit when a Stop, E-Stop, or Abort Event occurs.

This program demonstrates how to configure the DSP to toggle an I/O bit when a Stop, E-Stop, or
Abort Event occurs.  This is useful when external devices such as lasers or welding torches must
be disabled during exception events.

The Stop, E-Stop, and Abort Events are executed by the DSP based on an axis' limit switches, home
switch, software limits, error limits, etc.  Internally, the DSP executes some special frames to
complete the exception event.

When a Stop Event or E-Stop Event is generated, the DSP executes a Stop or E-Stop frame to decel-
erate the axis.  Then the DSP executes the Stopped frame to set the command velocity to zero.  An
Abort Event puts the axis in idle mode (no PID update) and executes the Stopped frame.  See the "C
Programming" manual for more information about Exception Events.

The frame offset addresses of the Stop, E-Stop, and Stopped frames are defined
 in IDSP.H:
#  define  EF_POS_STOP       0
#  define  EF_NEG_STOP       1
#  define  EF_POS_E_STOP     2
#  define  EF_NEG_E_STOP     3
#  define  EF_STOPPED        4

By modifying the Stop or E-Stop exception frames, a user I/O bit (or bits) can be toggled by the
DSP during the deceleration portion of a Stop or E-Stop.

By modifying the Stopped exception frame, a user I/O bit (or bits) can be toggled by the DSP after
a Stop, E-Stop, or Abort Event executes.

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
*/

/* Revision Control System Information
$Source$
$Revision$
$Date$
$Log$
*/

# include <stdio.h>
# include <stdlib.h>
# include <conio.h>

# include "pcdsp.h"
# include "idsp.h"

# define AXIS    0
# define PORT    0

# define BIT_NUMBER  0             /* Bits are numbered 0 to 47*/

# define VELOCITY    4000.0
# define ACCEL       40000.0
```

eventio.c

Toggle output bit when Stop/E-Stop/Abort event occurs

```
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;
   }
}

int16 display(int16 axis, int16 io_bit)
{
   int16 a_state, bit;
   double cmd, act;

   printf("\n");
   do
   {   get_position(axis, &act);
       get_command(axis, &cmd);
       a_state = axis_state(axis);
       bit = bit_on(io_bit);
       printf("Cmd %6.0lf Act %6.0lf State %d Bit %d\r", cmd, act, a_state, bit);

       if (kbhit())
       {   getch();
           return 0;
       }
   }
   while (!motion_done(axis));

   return 0;
}

void recover(int16 axis, int16 io_bit)
{
   display(axis, io_bit);
   error(controller_run(axis));
   reset_bit(io_bit);
}

int16 config_ef_io(PFRAME f, int16 port, int16 ormask, int16 andmask)
{
   int16 address,
       update = f->f.trig_update & 0xFF;           /* get trigger/update field */
   int16 * i = (int16*) &(f->f.position);          /* pointer to position field */

   if (!(update & FUPD_POSITION))
   {
       dsp_port_address(port, &address);           /* read the port address */
       f->f.output = OUTPUT_POSITION ;             /* set output to use position field */
       i[0] = address;
       i[1] = ormask;
       i[2] = andmask;
       f->f.trig_update |= FUPD_OUTPUT;            /* update I/O from position field */
       pcdsp_write_ef(dspPtr, f);                  /* download the exception frame */
       return DSP_OK ;
   }
   return DSP_RESOURCE_IN_USE;
}
```

```
int16 set_ef_io(int16 axis, int16 ef, int16 port, int16 ormask, int16 andmask)
{
   FRAME f;

   if (pcdsp_read_ef(dspPtr, &f, axis, ef))
       return dsp_error;

   if (config_ef_io(&f, port, ormask, andmask))
       return dsp_error;
   else
       return pcdsp_write_ef(dspPtr, &f);
}

int16 main()
{
   int16 error_code;

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

   init_io(PORT, IO_OUTPUT);
   set_io(PORT, 0x00);              /* set all 8 bits low */

   /* Configure the Stop Event frames to set an I/O bit high */
   error(set_ef_io(AXIS, EF_POS_STOP, PORT, 0x1, 0xFF));
   error(set_ef_io(AXIS, EF_NEG_STOP, PORT, 0x1, 0xFF));
   v_move(AXIS, VELOCITY, ACCEL);

   printf("\n\nPress a key to generate a Stop Event (with a bit toggle)");
   display(AXIS, BIT_NUMBER);
   set_stop(AXIS);
   recover(AXIS, BIT_NUMBER);

   /* Configure the E-Stop Event frames to set an I/O bit high */
   set_ef_io(AXIS, EF_POS_E_STOP, PORT, 0x1, 0xFF);
   set_ef_io(AXIS, EF_NEG_E_STOP, PORT, 0x1, 0xFF);
   v_move(AXIS, VELOCITY, ACCEL);

   printf("\n\nPress a key to generate an E-Stop Event (with a bit toggle)");
   display(AXIS, BIT_NUMBER);
   set_e_stop(AXIS);
   recover(AXIS, BIT_NUMBER);

   /* Configure the Stopped frame to set an I/O bit high.  The Stopped frame
       executes after a Stop, E-Stop, or Abort Event */
   set_ef_io(AXIS, EF_STOPPED, PORT, 0x1, 0xFF);
   v_move(AXIS, VELOCITY, ACCEL);

   printf("\n\nPress a key to generate an Abort Event (with a bit toggle)");
   display(AXIS, BIT_NUMBER);
   controller_idle(AXIS);           /* generate an Abort Event */
   recover(AXIS, BIT_NUMBER);

   /* Return frames to default configuration */
   error(dsp_reset());/* hardware reset */

   return 0;
}
```

Toggle output bit when Stop/E-Stop/Abort event occurs

eventrec.c

Recover from simple exception event

Exception Handling & Recovery

```
/* EVENTREC.C

:Simple Exception Event recovery

This sample demonstrates how to recover from an Exception Event.  The Exception Events can be gen-
erated by the DSP (dedicated digital I/O or software limits) or directly from software.  The
Events are Stop, Emergency Stop, and Abort.

The steps to exception event handling are:
 1) Determine what type of exception event occurred.
 2) Determine the cause of the exception event.
 3) Recover from the event based on the cause.

In this simple sample, the event recovery does not depend upon the cause of the event.  In a
"real" application the recovery routine may be different depending on the source of the event.

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
*/

/*  Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/

# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"

# define AXIS    0
# define VEL     1000.0
# define ACCEL   5000.0

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;
   }
}
```

```
void display(int16 axis)
{
   int16 pos_lim, neg_lim, state;
   double cmd;

   while (!motion_done(axis))
   {   get_command(axis, &cmd);
       printf("Cmd:%8.0lf\r", cmd);

       if (kbhit())
       {   getch();
           exit(1);
       }
   }
   printf("\n");
}

int16 check_for_event(int16 axis)
{
   int16 state;

   state = axis_state(axis);

   /* Check if an Exception Event occurred */
   switch (state)
   {
       case STOP_EVENT:
       {   printf("\nException Event = Stop");
           return state;
       }

       case E_STOP_EVENT:
       {   printf("\nException Event = Emergency Stop");
           return state;
       }

       case ABORT_EVENT:
       {   printf("\nException Event = Abort");
           return state;
       }
   }
   return 0;
}

int16 recover(int16 axis, int16 state)
{
   int16 recover_dir;

   /*  For this sample, the source of the event is reported to the screen and
        then the event is cleared:

        clear_status(...) - for STOP_EVENT or E_STOP_EVENT
        controller_run(...), enable_amplifier(...) - for ABORT_EVENT

       In a "real" application the recovery routine may be different depending
        on the source of the event.
   */

   switch (axis_source(axis))
   {
       case ID_HOME_SWITCH:
       {   printf("\nSource = Home Input");
           break;
       }

       case ID_POS_LIMIT:
       {   printf("\nSource = Positive Limit Input");
           break;
       }
```

```
        case ID_NEG_LIMIT:
        {   printf("\nSource = Negative Limit Input");
            break;
        }

        case ID_AMP_FAULT:
        {   printf("\nSource = Amp Fault Input");
            break;
        }

        case ID_X_NEG_LIMIT:
        {   printf("\nSource = Negative Software Limit Exceeded");
            break;
        }

        case ID_X_POS_LIMIT:
        {   printf("\nSource = Positive Software Limit Exceeded");
            break;
        }

        case ID_ERROR_LIMIT:
        {   printf("\nSource = Position Error Limit Exceeded");
            break;
        }
    }

    /* Make sure the event is complete before clearing the status. */
    while (!motion_done(axis))
        ;

    /* Note: Simply clearing the exception event (no matter what caused the
        event) may not be appropriate for all applications.
    */
    if (state == ABORT_EVENT)
    {   controller_run(axis);
        enable_amplifier(axis);
    }

    if ((state == E_STOP_EVENT) || (state == STOP_EVENT))
        error(clear_status(axis));

    return 0;
}

int16 main()
{
    int16 error_code, state;

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

    /* Configure the software limit */
    set_positive_sw_limit(AXIS, 10000.0, STOP_EVENT);

    printf("\nMoving to the positive software limit (any key to quit)\n\n");
    v_move(AXIS, VEL, ACCEL);
    display(AXIS);

    state = check_for_event(AXIS);
    if (state)
        recover(AXIS, state);

    return 0;
}
```

ext_int.c

Initialize -external interrupts under WinNT

Windows NT

```
/* EXT_INT.C

:Initialization for external interrupts under Windows NT.

This sample demonstrates how to initialize a DSP-Series controller to generate interrupts via User
I/O bit #23 under Windows NT.

This program is put to "sleep" until an interrupt is sent from the DSP-Series controller to the
host CPU.

When using interrupts, be sure to set the appropriate IRQ switch on the DSP-Series controller.
Also, make sure the device driver "DSPIO" is configured for the same IRQ.

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 <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "idsp.h"

#ifdef MEI_MSVC20           /* support for Microsoft Visual C/C++ ver 2.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif

#ifdef MEI_MSVC40           /* support for Microsoft Visual C/C++ ver 4.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif


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;
   }
}
```

ext_int.c

```
int main()
{
   int16 error_code;

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

   init_io(2, IO_INPUT);     /* initialize I/O port for inputs */

   /* Initialize DSP card for External Source Interrupts */
   dsp_interrupt_enable(TRUE);

   /* Wait for External Interrupt to occur */
   /* for MEI_TOGGLE and MEI_IO_MON axis is ignored, use 0.*/
   mei_sleep_until_interrupt(dspPtr->dsp_file, 0, MEI_TOGGLE);

   printf("External Interrupt Occurred!!!!!!!\n");
   printf("Hit any key to exit.\n");

   getch();

   return 0;
}
```

Initialize -external interrupts under WinNT

fbf.c

Perform feedback fault checking

Exception Handling & Recovery

```
/* FBF.C

:Feedback fault checking.

This sample demonstrates how to configure the DSP-Series controller to detect feedback fault con-
ditions.  Currently, broken wires and illegal states can be detected with differential incremental
encoders.

Feedback fault checking requires version 2.4F4 firmware (or higher) and the following hardware
revisions (or higher):

   Controller Rev

   PCX/DSP    3
   STD/DSP    8
   104/DSP    6
   LC/DSP     8
   V6U/DSP    3

Two registers determine the status of the encoder for each axis:

   Address    Status

   0x70       Broken Encoder Wire (each bit represents an axis)
   0x71       Illegal Encoder State (each bit represents an axis)

Every sample, the DSP will check these registers if the FB_CHECK (0x400) bit is set in the config
word for a given axis.  If a bit is set in either register (0x70 or 0x71), the DSP will generate
an Abort event corresponding to the axis at fault.

The type of encoder fault can be determined by reading the Broken wire and illegal state regis-
ters.  These registers can be cleared by writing a 0 to them.

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 Motion Library Version 2.5
Written for Firmware Version 2.4F4
*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "pcdsp.h"
#include "idsp.h"

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()
{
    int16 error_code, source, axis, axes;

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

    /* Enable feedback fault checking. */
    for (axis = 0; axis < PCDSP_MAX_AXES; axis++)
        set_feedback_check(axis, TRUE);

    printf("\nFeedback fault checking enabled (any key to quit).\n");

    while (!kbhit())
        printf("Fault: 0x%4X Illegal: 0x%4X\r", dsp_read_dm(FB_FAULT),
            dsp_read_dm(FB_ILLEGAL_STATE));
    getch();

    printf("\n\nAxis Source");
    for (axis = 0; axis < PCDSP_MAX_AXES; axis++)
    {   source = axis_source(axis);
        printf("\n %1d    0x%2X  ", axis, source);

        if (source == ID_FEEDBACK_FAULT)
            printf("  Fault!");
        if (source == ID_FEEDBACK_ILLEGAL_STATE)
            printf("  Illegal State!");
    }

    for (axis = 0; axis < PCDSP_MAX_AXES; axis++)
    {   set_feedback_check(axis, FALSE);
        controller_run(axis);
    }

     return 0;
}
```

fr_int.c

Initialization -frame interrupts under WinNT

Windows NT

```
/* FR_INT.C

:Initialization for frame interrupts under Windows NT.

This sample demonstrates how to initialize a DSP-Series controller to generate frame interrupts
under Windows NT.  Frames can be configured to generate an interrupt to the DSP by setting the
FCTL_INTERRUPT bit in the frame's control word with dsp_control(...).

This program is put to "sleep" until an interrupt is sent from the DSP-Series controller to the
host CPU.

When using interrupts, be sure to set the appropriate IRQ switch on the DSP-Series controller.
Also, make sure the device driver "DSPIO" is configured for the same IRQ.

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 <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "idsp.h"

#ifdef MEI_MSVC20           /* support for Microsoft Visual C/C++ ver 2.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif

#ifdef MEI_MSVC40           /* support for Microsoft Visual C/C++ ver 4.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif


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;
   }
}
```

fr_int.c

```
int main()
{
   int16 error_code;

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

   init_io(2, IO_OUTPUT);           /* initialize I/O port for outputs */
   reset_bit(23);                   /* set bit 23 LOW, to enable interrupts */

   /* Have the DSP generate an interrupt when an exception event occurs */
   interrupt_on_event(0, TRUE);

   start_move(0, 100000.0, 10000.0, 100000.0);
   dsp_control(0, FCTL_INTERRUPT, TRUE);           /* enable the frame interrupt bit */
   dsp_dwell(0, 1.0);                        /* download a dwell frame to generate an interrupt */
   dsp_control(0, FCTL_INTERRUPT, FALSE);
   dsp_end_sequence(0);                            /* end the frame sequence */
   start_move(0, 0.0, 10000.0, 100000.0);

   /* Wait for External Interrupt to occur.  To wait for a frame interrupt, use
       the define MEI_FRAME, to wait for an exception event, use MEI_EVENT. */
   /* for MEI_TOGGLE and MEI_IO_MON axis is ignored, use 0.*/
   mei_sleep_until_interrupt(dspPtr->dsp_file, 0, MEI_FRAME);

   printf("Frame Interrupt Generated!!!!!!\n");
   printf("Hit any key to exit.\n");

   getch();

   return 0;
}
```

fs.c

Simple feed speed control

Feed Speed Override

```
/* FS.C

:Simple feed speed control

This code commands the AXIS to move at a constant velocity.  The feed speed is adjusted for all of
the axes based on keyboard input.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"

# define  AXIS       0

# define  READY      0
# define  INCREASE   2
# define  DECREASE   3

# define  RATE_CHANGE .1

int16  state;
double feed_rate;

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;
   }
}
```

fs.c

```
void CheckState(void)
{
   switch (state)
   {
        case INCREASE:
        if (feed_rate < 2.0)
        { feed_rate += RATE_CHANGE;
             dsp_feed_rate(feed_rate);
          }
            state = READY;
            break;

        case DECREASE:
        if (feed_rate >= 0.0)
        { feed_rate -= RATE_CHANGE;
             dsp_feed_rate(feed_rate);
          }
            state = READY;
            break;

        case READY:
            break;
    }
}


int16 main()
{
   int16 error_code, done, key;
    double cmd, act, err;

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

    feed_rate = 1.0;
    v_move(AXIS, 1000.0, 10000.0);  /* accelerate to constant velocity */

    printf("\n +:increase, -:decrease, esc:quit\n");

   for (done = 0; !done; )
   {
       get_position(AXIS, &act);
       get_command(AXIS, &cmd);
       get_error(AXIS, &err);
       printf("CMD %6.0lf ACT %6.0lf ERR %6.0lf FEED %4.2lf\r", cmd, act, err, feed_rate);

       CheckState();

       if (kbhit())/* key pressed? */
       {   key = getch();

           switch (key)
           {
                 case '+':
                     state = INCREASE;
                     break;

                 case '-':
                     state = DECREASE;
                     break;

              case 0x1B:/* <ESC> */
                  v_move(AXIS, 0.0, 10000.0);
                     done = TRUE;
                  break;
           }
       }
   }
    return 0;
}
```

getdata.c

Read data buffer with device driver under WinNT

Windows NT

```
/* GETDATA.C

:Reads a data buffer with the device driver under Windows NT.

This sample demonstrates how to use the internal library function get_data_from_driver(...) under
Windows NT.  In this sample, the configuration word, internal offset, home port offset, home mask,
and home action for the first 3 axes is read.

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 <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "idsp.h"

#ifdef MEI_MSVC20           /* support for Microsoft Visual C/C++ ver 2.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif

#ifdef MEI_MSVC40           /* support for Microsoft Visual C/C++ ver 4.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif

#define AXES  3


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;
   }
}
```

getdata.c

```
int main()
{
   int16 error_code, *data, addr, data_offset, datum_len, pieces_of_data, axis;

   data = (int16*)calloc(AXES, 2*5);        /* allocate a buffer to hold the data */

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

   addr = dspPtr->e_data | PCDSP_DM;        /* beginning of config structs */
   data_offset = ED_SIZE;                   /* size of config struct */
   datum_len = 5;                           /* reading the first 5 words */
   pieces_of_data = AXES;                   /* read info for first 3 axes */
   get_data_from_driver(addr, data_offset, datum_len, pieces_of_data, data);

   for(axis = 0; axis < AXES; axis++)
   {   printf("Axis: %d\n", axis);
       printf("Config word: %d\n", data[axis*5]);
       printf("Internal Offset: %d\n", data[axis*5+1]);
       printf("Home Port Offset: %d\n", data[axis*5+2]);
       printf("Home Mask: %d\n", data[axis*5+3]);
       printf("Home Action: %d\n\n", data[axis*5+4]);
   }
   return 0;
}
```

Read data buffer with device driver under WinNT

holdit.c

Pause & resume motion on a path

Feed Speed Override

```
/* HOLDIT.C

:Pause and resume motion on path.

This sample demonstrates how to use the feed speed override to pause and continue motion on path.
To pause the motion the feed rate is reduced to zero.  To resume motion the feed rate is increased
to 100%.  To abort the motion, Stop Events are generated on the axes.

This sample is programmed as a state machine.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <conio.h>
# include <dos.h>
# include <stdlib.h>
# include "pcdsp.h"

int16 state;                /* one state variable for all axes */
double rate;                /* feed speed override rate */

/* State Values */
# define READY       0
# define HALT        1      /* pause a move */
# define GO          2      /* continue a move */
# define MOVE_FWD    3
# define MOVE_BACK   4
# define NEVER_MIND  5      /* abort the motion */

# define AXES    2


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;
    }
}

void display(int16 n_axes, int16 * map)
{
    int16 i;
    double cmd;

    for (i = 0; i < n_axes; i++)
    {  get_command(map[i], &cmd);
        printf("%d: %10.0lf ", map[i], cmd);
    }
    printf("\r");
}

void CheckState(int16 n_axes, int16 * map)
{
   int16 i;

   switch (state)
   {
       case HALT:
           rate -= .01;
           if(rate <= 0)
           {  rate = 0;
              state = READY;
           }
           error(dsp_feed_rate(rate));/* update feed rate */
           break;

       case GO:
           rate += .01;
           if(rate >= 1)
           {  rate = 1;
              state = READY;
           }
           error(dsp_feed_rate(rate));     /* update feed rate */
           break;

       case MOVE_FWD:
           move_2(100000.0, 200000.0);
           state = READY;
           break;

       case MOVE_BACK:
           move_2(0.0, 0.0);
           state = READY;
           break;

       case NEVER_MIND:              /* discontinue the move */
           if(rate == 0.0)
           {  for (i = 0; i < n_axes; i++)
              {   set_jerk(map[i], 0.0);           /* zero the trajectory generator */
                  set_accel(map[i], 0.0);
                  set_velocity(map[i], 0.0);
                  stop_motion();                   /* generate Stop Events */
                  rate = 1.0;
                  error(dsp_feed_rate(rate));      /* execute the Stop Events */
              }
           }
           else
              stop_motion();

           while(!all_done())
              ;
           for (i = 0; i < n_axes; i++)
              error(clear_status(map[i]));
```

```
            printf("\n");
            state = READY;
            break;
    }
}

int16 main()
{
    int16 error_code, done, key, n_axes, axes[] = {0, 1};

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

    n_axes = (sizeof(axes) / sizeof(int16));

    map_axes(n_axes, axes);
    set_move_speed(8000.0);
    set_move_accel(80000.0);

    printf("\nf=fwd, b=back, h=halt, g=go, n=endmove, esc=quit\n");

    for (done = 0; !done; )
    {
        display(n_axes, axes);
        CheckState(n_axes, axes);

        if (kbhit())          /* key pressed? */
        {   key = getch() ;

            switch (key)
            {
            case 'h':         /* Pause the motion */
                state = HALT;
                break;

            case 'g':         /* Resume the motion */
                state = GO;
                break;

            case 'f':         /* Move the axes forward */
                state = MOVE_FWD;
                break;

            case 'b':         /* Move the axes backward */
                state = MOVE_BACK;
                break;

            case 'n':         /* End the motion */
                state = NEVER_MIND;
                break;

            case 0x1B:        /* <ESC> */
                done = TRUE;
                break;
            }
        }
    }
    return 0;
}
```

home1.c

Homing: Simple, using home input

Homing Routines

```
/* HOME1.C

:Simple homing routine using the home input

This sample demonstrates a basic homing algorithm.  The home location is found based on the home
input.

Here is the algorithm:
 1) Velocity move towards the home input
 2) Stop at the home sensor (Stop Event generated by DSP)
 3) Zero the position

Here is the sensor logic:
 Home input   = active high
 Index input  = not used

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
*/

/*  Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/

# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"

# define AXIS        0
# define FAST_VEL    10000.0
# define ACCEL       50000.0

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;
   }
}
```

```
void display(int16 axis)
{
   int16 home_logic, state;
   double cmd;

   while (!motion_done(axis))
   {   get_command(axis, &cmd);
       printf("\rCmd: %10.0lf Home Logic: %d", cmd, home_switch(AXIS));

       if (kbhit())
       {   getch();
           exit(1);
       }
   }
   printf("\n");
}

int16 main()
{
   int16 error_code;

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

   set_home_index_config(AXIS, HOME_ONLY);
   set_home_level(AXIS, TRUE);      /* home input logic active high */
   set_home(AXIS, STOP_EVENT);
   set_stop_rate(AXIS, ACCEL);

   /* Clear any events generated during home logic configuration */
   error(clear_status(AXIS));

   printf("\nSearching for home sensor. (Press any key to quit)\n");
   v_move(AXIS, FAST_VEL, ACCEL);   /* velocity move towards home sensor */
   display(AXIS);

   set_home(AXIS, NO_EVENT);
   error(clear_status(AXIS));
   set_position(AXIS, 0.0);         /* zero command and actual position */
   printf("\nAxis is home.\n");

   return 0;
}
```

home2.c

Homing: Two-stage homing, using home input

Homing Routines

```
/* HOME2.C

:Two stage homing routine using the home input

This sample demonstrates a basic homing algorithm.  The home location is found
based on the home input.

Here is the algorithm:
 1) Fast velocity move towards the home sensor
 2) Stop at the home sensor (Stop Event generated by DSP)
 3) Position move off the home sensor (opposite direction)
 4) Slow velocity move towards the home sensor
 5) Stop at the home sensor (Stop Event generated by DSP)
 6) Zero the position.

Here is the sensor logic:
 Home input   = active high
 Index input  = not used

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
*/

/*  Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/

# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"

# define AXIS        0
# define SLOW_VEL    200.0
# define FAST_VEL    20000.0
# define SLOW_ACCEL  10000.0
# define FAST_ACCEL  50000.0

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;
   }
}

void display(int16 axis)
{
   int16 home_logic, state;
   double cmd;

   while (!motion_done(axis))
   {   get_command(axis, &cmd);
       printf("\rCmd: %10.0lf Home Logic: %d", cmd, home_switch(AXIS));

       if (kbhit())
       {   getch();
           exit(1);
       }
   }
   printf("\n");
}

int16 main()
{
   int16 error_code;
   double distance;

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

   /* make sure no events are generated during home logic configuration */
   set_home(AXIS, NO_EVENT);
   error(clear_status(AXIS));

   set_stop_rate(AXIS, SLOW_ACCEL);
   set_home_index_config(AXIS, HOME_ONLY);
   set_home_level(AXIS, TRUE);
   set_home(AXIS, STOP_EVENT);

   printf("\nQuickly searching for home sensor. (Press any key to quit)\n");
   v_move(AXIS, FAST_VEL, SLOW_ACCEL);      /* velocity move towards home sensor */
   display(AXIS);

   set_home(AXIS, NO_EVENT);
   error(clear_status(AXIS));

   /* move off by a dist equal to twice the decel distance */
   distance = FAST_VEL * FAST_VEL / SLOW_ACCEL;
   start_r_move(AXIS, -distance, FAST_VEL, FAST_ACCEL);
   display(AXIS);

   printf("\nSlowly searching for home sensor. (Press any key to quit)\n");
   set_stop_rate(AXIS, FAST_ACCEL);
   set_home(AXIS, STOP_EVENT);
   v_move(AXIS, SLOW_VEL, FAST_ACCEL);
   display(AXIS);

   set_position(AXIS, 0.0);         /* zero command and actual position */
   printf("\nAxis is home.\n");

   return 0;
}
```

home3.c

Homing: Using combined home & index logic

Homing Routines

```
/* HOME3.C

:Homing routine using the combined home and index logic

This sample demonstrates a basic homing algorithm.  The home location is found based on the home
and index input logic.

Here is the algorithm:
 1) Velocity move towards the home input anded with the index
 2) Decel to stop at home and index
   (New Frame Event generated by DSP)
 3) Calculate where the home/index occured
 4) Move to the home position
 5) Zero home position

Here is the sensor logic:
 Home input   = active high
 Index input  = active high

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
*/

/*  Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/

# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"

# define AXIS        0
# define VELOCITY    500.0
# define ACCEL       10000.0

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;
   }
```

```
}

void display(int16 axis)
{
   double cmd;

   while (!motion_done(axis))
   {   get_command(axis, &cmd);
       printf("\rCmd: %10.0lf Home Logic: %d", cmd, home_switch(AXIS));

       if (kbhit())
       {   getch();
           exit(1);
       }
   }
   printf("\n");
}

int16 main()
{
   int16 error_code;

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

   /* make sure no events are generated during home logic configuration */
   set_home(AXIS, NO_EVENT);
   error(clear_status(AXIS));

   set_stop_rate(AXIS, ACCEL);
   set_home_index_config(AXIS, HIGH_HOME_AND_INDEX);
   set_home_level(AXIS, FALSE);

   printf("\nSearching for home and index. (Press any key to quit)\n");
   v_move(AXIS, VELOCITY, ACCEL);           /* velocity move towards home sensor */
   /* Configure dsp to generate a new frame when home input occurs. */
   dsp_home_action(AXIS, NEW_FRAME);
   dsp_dwell(AXIS, 10000.0);                /* wait for New frame event */
   dsp_home_action(AXIS, NO_EVENT);
   set_position(AXIS, 0.0);                 /* stop when home event occurs */
   v_move(AXIS, 0.0, ACCEL);
   display(AXIS);

   /* Calculate the time it takes to decel to a stop */
   time = (VELOCITY/ACCEL) ;
   /* Calculate distance moved during deceleration. */
   decel offset = .5 * ACCEL * time * time
   /* Calculate the distance attributed to a single dsp sample for accuracy */
   sample offset = VELOCITY * (1/dsp sample_rate()) ;
   /* Total distance moved during decel */
   home_pos = decel_offset + sample_offset;

   start r move(AXIS, -home_pos, VELOCITY, ACCEL);/* move to home position */
   display(AXIS);

   set position(AXIS, 0.0) ;                /* zero command and actual position */
   printf("\nAxis is home.\n");

   return 0;
}
```

home4.c

Homing: Using home & index inputs

Homing Routines

```
/* HOME4.C

:Homing routine using the home and index inputs

This sample demonstrates a basic homing algorithm.  The home location is found based on the home
and index input logic.

Here is the algorithm:
 1) Velocity move towards the home input
 2) Stop at the home sensor (Stop Event generated by DSP)
 3) Velocity move off the home sensor (opposite direction)
 4) Zero the position at the index pulse (New Frame Event generated by DSP)
 5) Stop the axis
 6) Move to the home position

Here is the sensor logic:
 Home input   = active high
 Index input  = active high

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
*/

/*  Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/

# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"

# define AXIS        0
# define FAST_VEL    20000.0
# define SLOW_VEL    500.0
# define ACCEL       10000.0

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;
```

```
   }
}

void display(int16 axis)
{
   int16 home_logic, state;
   double cmd;

   while (!motion_done(axis))
   {   get_command(axis, &cmd);
       printf("\rCmd: %10.0lf Home Logic: %d", cmd, home_switch(AXIS));

       if (kbhit())
       {   getch();
           exit(1);
       }
   }
   printf("\n");
}

int16 main()
{
   int16 error_code;

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

   /* make sure no events are generated during home logic configuration */
   set_home(AXIS, NO_EVENT);
   error(clear_status(AXIS));

   set_stop_rate(AXIS, ACCEL);
   set_home_index_config(AXIS, HOME_ONLY);
   set_home_level(AXIS, TRUE);
    set_home(AXIS, STOP_EVENT);

   printf("\nSearching for home sensor. (Press any key to quit)\n");
   v_move(AXIS, FAST_VEL, ACCEL);          /* velocity move towards home sensor */
   display(AXIS);

   set_home(AXIS, NO_EVENT);
   set_home_index_config(AXIS, INDEX_ONLY);
   error(clear_status(AXIS));

   printf("\nSearching for index. (Press any key to quit)\n");
   v_move(AXIS, -SLOW_VEL, ACCEL);         /* velocity move off home sensor */
   dsp_home_action(AXIS, NEW_FRAME);
   dsp_dwell(AXIS, 10000.0);               /* wait for New frame event */
   dsp_home_action(AXIS, NO_EVENT);
   set_position(AXIS, 0.0);                /* zero command and actual position */
   v_move(AXIS, 0.0, ACCEL);

   display(AXIS);
   start_move(AXIS, 0.0, FAST_VEL, ACCEL);  /* move to home location */
   display(AXIS);
   printf("\nAxis is home.\n");

   return 0;
}
```

home5.c

Homing: Using a mechanical hard stop

Homing Routines

```
/* HOME5.C

:Homing routine using a mechanical hard stop.

This sample demonstrates a basic homing algorithm.  The home location is found based on a mechan-
ical hard stop.  The DSP senses the hard stop when the difference between the command and actual
positions exceeds the error limit.

Here is the algorithm:
 1) Velocity move towards the hard stop
 2) Stop at the mechanical hard stop (when the error limit is exceeded)
 3) Zero the command and actual position

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
*/

/*  Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/

# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"

# define AXIS        0
# define VEL         1000.0
# define ACCEL       1000.0
# define FAST_ACCEL  100000.0

# define BACK_OFF_DIST      (-3000.0)
# define HOME_ERR_LIMIT     20.0
# define HOME_DAC_LIMIT     3276           /* limits control voltage to +/- 1 volt */

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;
   }
}
```

```
void display(int16 axis)
{
   double cmd, act;

   while (!motion_done(axis))
   {   get_command(axis, &cmd);
       get_position(axis, &act);
       printf("\rCmd: %10.0lf Act: %10.0lf", cmd, act);
   }
   printf("\n");
}

int16 main()
{
   int16 error_code, coeffs[COEFFICIENTS], dac_limit, err_event;
   double err_limit;

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

   get_filter(AXIS, coeffs);
   get_error_limit(AXIS, &err_limit, &err_event);

   dac_limit = coeffs[DF_DAC_LIMIT];        /* reduce the dac limit */
   coeffs[DF_DAC_LIMIT] = HOME_DAC_LIMIT;
   set_filter(AXIS, coeffs);

   printf("\nSearching for mechanical limit.\n");
   set_error_limit(AXIS, HOME_ERR_LIMIT, NO_EVENT);
   v_move(AXIS, VEL, ACCEL);                /* velocity move towards hard stop */
   dsp_error_action(AXIS, NEW_FRAME);
   dsp_dwell(AXIS, 100000.0);               /* wait for New Frame event */
   dsp_error_action(AXIS, NO_EVENT);
   v_move(AXIS, 0.0, FAST_ACCEL);           /* decel to a stop */
   set_position(AXIS, 0.0);                 /* zero command and actual position */
   dsp_end_sequence(AXIS);
   display(AXIS);

   printf("\nAxis is home. Hit any key to back off home.\n");
   getch();
   start_r_move(AXIS, BACK_OFF_DIST, VEL, ACCEL);
   display(AXIS);

   coeffs[DF_DAC_LIMIT] = dac_limit;        /* reset the dac limit */
   set_filter(AXIS, coeffs);
   set_error_limit(AXIS, err_limit, err_event);

   return 0;
}
```

home6.c

Homing: Using encoder's index pulse

Homing Routines

```
/* HOME6.C

:Simple homing routine using the encoders' index pulse.

This sample demonstrates a basic homing algorithm.  The home location is found based on the encod-
ers' index pulse.

Here is the algorithm:
 1) Velocity move towards the index pulse
 2) Stop at the index pulse (Stop Event generated by DSP)
 3) Zero the position

Here is the sensor logic:
 Index input     = active high
 Home input      = not used

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
*/

/*  Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/

# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"

# define AXIS       0
# define ACCEL      50000.0
# define HOME_VEL   1000.0         /* Velocity must be less than the DSP's
                                      sample rate to guarantee the index
                                      pulse is not missed.              */


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;
   }
}
```

```
void display(int16 axis)
{
   int16 home_logic, state;
   double cmd;

   while (!motion_done(axis))
   {
       get_command(axis, &cmd);
       printf("\rCmd: %10.0lf Home Logic: %d", cmd, home_switch(AXIS));

       if (kbhit())
       {   getch();
           exit(1);
       }
   }
   printf("\n");
}

int16 main()
{
   int16 error_code;

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

   set_home_index_config(AXIS, INDEX_ONLY);
   set_home_level(AXIS, TRUE);         /* home input logic active high */
   set_home(AXIS, STOP_EVENT);
   set_stop_rate(AXIS, ACCEL);

   /* Clear any events generated during home logic configuration */
   error(clear_status(AXIS));

   printf("\nSearching for index pulse. (Press any key to quit)\n");
   v_move(AXIS, HOME_VEL, ACCEL);      /* velocity move towards index pulse */
   display(AXIS);

   set_home(AXIS, NO_EVENT);
   error(clear_status(AXIS));
   set_position(AXIS, 0.0);            /* zero command and actual position */
   printf("\nAxis is home.\n");

   return 0;
}
```

home7.c

Homing: Find midpoint between +/- limits

Homing Routines

```
/* HOME7.C

:Homing routine to find midpoint between positive and negative limits.

Here is the algorithm:
 1) Move to negative limit, store position
 2) Move to positive limit, store position
 3) Calculate total width of system
 4) Set position referenced from zero (middle position)

Here is the sensor logic:
 Home input      = not used
 Index input     = not used
 +Limit input    = active high
 -Limit input    = active high

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
*/

/*  Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include <math.h>
# include "pcdsp.h"

# define    AXIS        0
# define    POS_VEL     10000.0
# define    NEG_VEL     (-10000.0)
# define    ACCEL       50000.0


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;
```

```
   }
}


void display(int16 axis)
{
   int16 home_logic, state;
   double cmd;

   while (!motion_done(axis))
   {
       get_command(axis, &cmd);
       printf("\rCmd: %10.0lf Positive Limit: %d Negative Limit: %d", cmd, pos_switch(AXIS),
neg_switch(AXIS));


       if (kbhit())
       {   getch();
           exit(1);
       }
   }
   printf("\n");
}


int16 initialize_limits(int16 axis)
{
   set_positive_limit(axis, STOP_EVENT);
   set_positive_level(axis, TRUE);

   set_negative_limit(axis, STOP_EVENT);
   set_negative_level(axis, TRUE);

   return 0;
}


int16 find_neg_limit(int16 axis, double * min_pos)
{
   v_move(axis, NEG_VEL, ACCEL);            /* move to negative limit */
   printf("Moving to negative limit...\n");
   display(axis);
   get_position(axis, min_pos);
   error(clear_status(axis));

   return 0;
}


int16 find_pos_limit(int16 axis, double * max_pos)
{
   v_move(axis, POS_VEL, ACCEL);            /* move to positive limit */
   printf("Moving to positive limit...\n");
   display(axis);
   get_position(axis, max_pos);
   error(clear_status(axis));

   return 0;
}
```

home7.c

```
int16 main()
{
   int16 error_code;
   double min_pos, max_pos, width;

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

   initialize_limits(AXIS);                /* configure limit switches */

   find_neg_limit(AXIS, &min_pos);
   find_pos_limit(AXIS, &max_pos);
   width = (fabs(min_pos) + fabs(max_pos));
   printf("Width of system: %lf \n",width);

   /* reference position from middle (zero) of system */

   set_position(AXIS, (width / 2.0));       /* position at positive limit */
   printf("Current Position: %lf\n", (width / 2));

   return 0;
}
```

Homing: Find midpoint between +/- limits

inthnd.c

Single-board interrupt support

Interrupt Handling

```
/*  INTHND.C

:Sample code for Single-Board Interrupt Support.

This code demonstrates how to handle interrupts generated from a single DSP-Series controller.

When programming with interrupt routines, make sure to define CRITICAL and ENDCRITICAL.  These are
located in idsp.h.  CRITICAL is used to disable interrupts and ENDCRITICAL re-enables interrupts.
This is very important! Interrupts during communication can cause strange problems.

If you are using a Borland C compiler (3.1 or greater), then CRITICAL and ENDCRITICAL are defined
for you.

When compiling code with an interrupt routine, to turn stack checking off.

If you modify dsp_interrupt(...), watch out for floating point operations- most compilers do not
support them.  Also, MEI's standard function library is not supported inside interrupt routines.

Written for Borland and Microsoft compilers.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <dos.h>
# include <conio.h>
# include "idsp.h"

# define  R_8259_EOI        0x20
# define  R_8259_IMR        0x21
# define  EOI               0x20
# define  EF_NONE           (-1)

typedef void (interrupt * INTERRUPT_HANDLER)();

static int16 Read_DSP(unsigned addr);
static void Write_DSP(unsigned addr, int16 dm);
void install_dsp_handler(int16 intno);
typedef void (interrupt * INTERRUPT_HANDLER)();
INTERRUPT_HANDLER old_handler = NULL;
static int16 imr, interrupt_number;

int16 axis_interrupts[PCDSP_MAX_AXES];
int16 io_interrupts;
```

```
static int16 inc_acks(int16 axis)
{
    int16 addr1,addr2;
    int16 v1, v2 ;

    addr1 = dspPtr->global_data + axis + GD_SIZE;          /* irq_count */
    addr2 = addr1 + dspPtr->axes;                          /* ack_count */
    v1 = Read_DSP(addr1);
    v2 = Read_DSP(addr2);

    if (v1 != v2)                  /* Is the DSP generating interrupts? */
       Write_DSP(addr2,v1);        /* set ack_count = irq_count */
    return (v1 - v2);
}

void _far interrupt dsp_interrupt()
{
   int16 i;

   _disable();

   /* Is the interrupt caused by the DSP I/O monitoring? */
   if (Read_DSP((Read_DSP(DM_IO_BLOCK)) + IO_CHANGE))
   {
       io_interrupts++;
       /* Clear the I/O monitoring flag. */
       Write_DSP((Read_DSP(DM_IO_BLOCK) + IO_CHANGE), 0);
   }

   for (i = 0; i < PCDSP_MAX_AXES; i++)
   {
       if (inc_acks(i))
       {
           axis_interrupts[i]++;
       }
   }

    outp(R_8259_EOI, EOI);
}

void remove_handler(void)
{
   _disable() ;
   _dos_setvect(interrupt_number + 8, old_handler) ;
   outp(R_8259_IMR, imr) ;
   outp(R_8259_EOI, EOI) ;
   _enable() ;

}

void install_dsp_handler(int16 intno)
{
   int16 new_imr;

   interrupt_number = intno ;
   old_handler = _dos_getvect(interrupt_number + 8) ;
   imr = inp(R_8259_IMR) ;

   atexit(remove_handler) ;

   new_imr = imr & ~(1 << interrupt_number) ;
   _disable() ;
   _dos_setvect(interrupt_number + 8, dsp_interrupt) ;
   outp(R_8259_IMR, new_imr) ;
   outp(R_8259_EOI, EOI) ;
   _enable() ;
}
```

```
int16 dsp_irq_frame(int16 axis)
{
   FRAME frame ;
   frame_clear(&frame)  ;
   if (frame_allocate(&frame, dspPtr, axis))
       return dsp_error ;
   frame.f.control |= FCTL_INTERRUPT ;
   return frame_download(&frame) ;
}

static int16 Read_DSP(unsigned addr)
{  outpw(dspPtr->address,addr | 0x8000);
   return(inpw(dspPtr->data));
}

static void Write_DSP(unsigned addr, int16 dm)
{
   outpw(dspPtr->address,addr | 0x8000);
   outpw(dspPtr->data,dm);
}
```

intsimpl.c

Single-board interrupt support under DOS

Interrupt Handling

```
/*  INTSIMPL.C

:Sample code for Single-Board Interrupt Support under DOS.

This code demonstrates how to handle interrupts generated from  a single DSP-Series controller.

When programming with interrupt routines, make sure to define CRITICAL and ENDCRITICAL.  These are
located in idsp.h.  CRITICAL is used to disable interrupts and ENDCRITICAL re-enables interrupts.
This is very important!  Interrupts during communication can cause  strange problems.

If you are using a Borland C compiler (3.1 or greater), then CRITICAL  and ENDCRITICAL are defined
for you.

When compiling code with an interrupt routine, be sure to turn stack checking off.

This program uses a general method for installing and removing interrupt handlers.  The handler
itself is written specifically for this program and can easily be modified.

If you modify dsp_interrupt(...), watch out for floating point operations- most compilers do not
support them.  Also, MEI's standard function library is not supported inside interrupt routines.

Written for Borland and Microsoft compilers.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/

#  include <stdio.h>
#  include <stdlib.h>
#  include <dos.h>
#  include <conio.h>
#  include "idsp.h"

#  define  R_8259_EOI        0x20
#  define  R_8259_IMR        0x21
#  define  EOI               0x20

typedef void (interrupt * INTERRUPT_HANDLER)();
void install_dsp_handler(int16 intno);
INTERRUPT_HANDLER old_handler = NULL;
static int16 imr, interrupt_number;

int16 INTERRUPT_FLAG = FALSE;
```

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

    if(error_code)
   {   error_msg(error_code, buffer);
       printf("ERROR: %s\n", buffer);
         exit(1);
    }
}

void _far interrupt dsp_interrupt()
{
   int16 i;

   _disable();
   INTERRUPT_FLAG = TRUE;

   outp(R_8259_EOI, EOI);
   _enable() ;
}

void remove_handler(void)
{
   _disable() ;
   _dos_setvect(interrupt_number + 8, old_handler) ;
   outp(R_8259_IMR, imr) ;
   outp(R_8259_EOI, EOI) ;
   _enable() ;
}

void install_dsp_handler(int16 intno)
{
   int16 new_imr;

   interrupt_number = intno ;
   old_handler = _dos_getvect(interrupt_number + 8) ;
   imr = inp(R_8259_IMR) ;

   atexit(remove_handler) ;

   new_imr = imr & ~(1 << interrupt_number) ;
   _disable() ;
   _dos_setvect(interrupt_number + 8, dsp_interrupt) ;
   outp(R_8259_IMR, new_imr) ;
   outp(R_8259_EOI, EOI) ;
   _enable() ;
}

int16 main(void)
{
   int16 error_code;

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

   // Initialize port 2 for output.  All bits including bit 23 will be low.
   init_io(2, IO_OUTPUT);

   // Enable the DSP card to send interrupts to the PC.
   dsp_interrupt_enable(TRUE);

   // Install the interrupt handler.
   install_dsp_handler(5);

   // The INTERRUPT_FLAG should be 0 before the interrupt is sent.
   while(!kbhit())
      if(INTERRUPT_FLAG)
         printf("Interrupt received    \r");
      else
```

```
            printf("Interrupt not received\r");
      getch();

      // Send an interrupt by setting bit 23 HIGH.
      set_bit(23);

      // The INTERRUPT_FLAG should be 1 after the interrupt is sent.
      while(!kbhit())
         if(INTERRUPT_FLAG)
            printf("Interrupt received    \r");
         else
            printf("Interrupt not received\r");
      getch();

      return 0;
   }
```

iolatch.c

Latch actual positions of all axes, using User I/O bit 22

Position Latching

```
/* IOLATCH.C

:Latch the actual positions of all axes based on user I/O bit #22.

Be sure user I/O bit #22 is normally driven high, and is pulled low to activate the latch.  The
falling edge of bit #22 triggers the DSP's interrupt.  The DSP's interrupt routine handles the
latching of the actual positions of all axes within 4 microseconds.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"


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;
   }
}

int16 main()
{
   int16 error_code, axis, axes, done = 0;
   double p;

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

   axes = dsp_axes();
   printf("\nToggle bit #22 to latch %d axes, Press any key to quit\n", axes);
```

iolatch.c

```
    init_io(2, IO_INPUT);
    arm_latch(TRUE);

    while (!done)
    {
        while (!latch_status() && !done)
        {
            if (kbhit())
               done = getch();
        }
        for (axis = 0; axis < axes; axis++)
        {
            get_latched_position(axis, &p);
            printf("Axis:%d Position:%12.0lf\n", axis, p);
        }
        printf("\n");
        arm_latch(TRUE);      /* reset the latch. */
    }

     return 0;
}
```

Latch actual positions of all axes, using User I/O bit 22

iomon_in.c

Initialize for I/O-generated interrupts under WinNT

Windows NT

```
/* IOMON_IN.C

:Initialization for I/O generated interrupts under Windows NT.

This sample demonstrates how to initialize a DSP-Series controller to monitor I/O lines and gen-
erate interrupts under Windows NT.

This program is put to "sleep" until an interrupt is sent from the DSP-Series controller to the
host CPU.

When using interrupts, be sure to set the appropriate IRQ switch on the DSP-Series controller.
Also, make sure the device driver "DSPIO" is configured for the same IRQ.

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 <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "idsp.h"

#ifdef MEI_MSVC20           /* support for Microsoft Visual C/C++ ver 2.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif

#ifdef MEI_MSVC40           /* support for Microsoft Visual C/C++ ver 4.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif


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()
{
   int16 error_code;

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

   init_io(0, IO_INPUT);            /* initialize I/O port for inputs */
   init_io(1, IO_INPUT);

   init_io(2, IO_OUTPUT);           /* initialize I/O port for outputs */
   reset_bit(23);                   /* enable interrupt generation */

   /* initialize I/O Monitoring */
   set_io_mon_mask(0, 1, 0);
   clear_io_mon();

   /* Wait for I/O Monitoring to send an Interrupt */
   /* for MEI_TOGGLE and MEI_IO_MON axis is ignored, use 0.*/
   mei_sleep_until_interrupt(dspPtr->dsp_file, 0, MEI_IO_MON);

   printf("io monitor interrupt!!!!!!!\n");
   printf("Hit any key to exit.\n");

   getch();

   return 0;
}
```

jog.c

Use the jog_axes(...) function

Jogging

```
/* JOG.C

:Demonstrates the jog_axis(...) function

This code initializes 3 analog input channels for unipolar (0 to 5 volt) operation.  Then the DSP
is configured for velocity-based jogging.

The velocity is calculated as follows:

Each sample, the DSP runs through the following calculations to determine the velocity that the
motor will run at.

      if A/D value > (center + deadband)
          then J = A/D value - center - deadband
      if A/D value < (center - deadband)
          then J = A/D value - center + deadband
      if (center + deadband) > A/D value > (center - deadband)
          then J = 0

      motor velocity (counts/sec) =
(linear term * 1/65536 * J + cubic term * 3.632E-12 * (J * J * J)) * sample_rate

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/

# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"

# define  X   0
# define  Y   1
# define  Z   2

# define  XJOG    0
# define  YJOG    1
# define  ZJOG    2

# define  XANALOGCHANNEL    0
# define  YANALOGCHANNEL    1
# define  ZANALOGCHANNEL    2

# define  XCENTER    2048
# define  YCENTER    2048
# define  ZCENTER    2048

# define  XDEADBAND         5
```

```
# define  YDEADBAND         20
# define  ZDEADBAND         50

# define  XLINEAR    10
# define  YLINEAR    100
# define  ZLINEAR    1000

# define  XCUBIC     10
# define  YCUBIC     50
# define  ZCUBIC     100


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;
   }
}

int16 main()
{
   int16 error_code;

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

   /* Initialize X, Y, and Z channels to use specific analog channels and
   configure these channels to be unipolar and single ended.
   NOTE: the default analog channel for an axis is the same as the axis
   number and is configured for unipolar and single ended.
   */

   set_analog_channel(X, XANALOGCHANNEL, FALSE, FALSE);
   set_analog_channel(Y, YANALOGCHANNEL, FALSE, FALSE);
   set_analog_channel(Z, ZANALOGCHANNEL, FALSE, FALSE);

   jog_axis(X, XJOG, XCENTER, XDEADBAND, XLINEAR, XCUBIC, TRUE);
   jog_axis(Y, YJOG, YCENTER, YDEADBAND, YLINEAR, YCUBIC, TRUE);
   jog_axis(Z, ZJOG, ZCENTER, ZDEADBAND, ZLINEAR, ZCUBIC, TRUE);

   return 0;
}
```

joglat1.c

Jog/latch positions, generate stop events, clear status on 3 axes

Jogging

```
/* JOGLAT1.C

:Jog, latch positions, generate stop events, and clear status on three axes.

This code initializes 3 analog input channels for unipolar (0 to 5 volt) operation.  Then the DSP
is configured for velocity-based jogging. When positions are latched (by toggling User I/O bit
#22), a sequence of frames on a phantom axis are executed. These frames generate STOP_EVENTs on
axes (0-2). The status of each axis is then cleared to allow any further motion to be programmed.

The jogging velocity is calculated as follows:

   Each sample, the DSP runs through the following calculations to
   determine the velocity that the motor will run at.

       if A/D value > (center + deadband)
           then J = A/D value - center - deadband
       if A/D value < (center - deadband)
           then J = A/D value - center + deadband
       if (center + deadband) > A/D value > (center - deadband)
           then J = 0

       motor velocity (counts/sec) =
  (linear term * 1/65536 * J + cubic term * 3.632E-12 * (J * J * J)) * sample_rate

Be sure user I/O bit #22 is normally driven high, and is pulled low to activate the latch.  The
falling edge of bit #22 triggers the DSP's interrupt.  The DSP's interrupt routine handles the
latching of the actual positions of all axes within 4 microseconds.

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
*/

/*  Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"


# define  X   0
# define  Y   1
# define  Z   2

# define  XJOG    0
# define  YJOG    1
# define  ZJOG    2

# define  XANALOGCHANNEL     0
```

```
# define  YANALOGCHANNEL    1
# define  ZANALOGCHANNEL    2

# define  XCENTER    2048
# define  YCENTER    2048
# define  ZCENTER    2048

# define  XDEADBAND  5
# define  YDEADBAND  20
# define  ZDEADBAND  50

# define  XLINEAR    10
# define  YLINEAR    100
# define  ZLINEAR    1000

# define  XCUBIC     10
# define  YCUBIC     50
# define  ZCUBIC     100


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;
   }
}

void begin_jogging(void)
{
   /* Initialize X, Y, and Z channels to use specific analog channels and
   configure these channels to be unipolar and single ended.
   NOTE: the default analog channel for an axis is the same as the axis
   number and is configured for unipolar and single ended.
   */

   set_analog_channel(X, XANALOGCHANNEL, FALSE, FALSE);
   set_analog_channel(Y, YANALOGCHANNEL, FALSE, FALSE);
   set_analog_channel(Z, ZANALOGCHANNEL, FALSE, FALSE);

   jog_axis(X, XJOG, XCENTER, XDEADBAND, XLINEAR, XCUBIC, TRUE);
   jog_axis(Y, YJOG, YCENTER, YDEADBAND, YLINEAR, YCUBIC, TRUE);
   jog_axis(Z, ZJOG, ZCENTER, ZDEADBAND, ZLINEAR, ZCUBIC, TRUE);
}

void stop_with_latch(void)
{
   dsp_io_trigger(0,22,FALSE);  /* Execute next frame when bit 22 goes low */
   dsp_io_trigger(1,22,FALSE);
   dsp_io_trigger(2,22,FALSE);

   dsp_axis_command(0,0,STOP_EVENT);
   dsp_axis_command(1,1,STOP_EVENT);
   dsp_axis_command(2,2,STOP_EVENT);
}
```

```
void recover_from_latch(void)
{
   /* Disable jogging on three axes */

   jog_axis(X, XJOG, XCENTER, XDEADBAND, XLINEAR, XCUBIC, FALSE);
   jog_axis(Y, YJOG, YCENTER, YDEADBAND, YLINEAR, YCUBIC, FALSE);
   jog_axis(Z, ZJOG, ZCENTER, ZDEADBAND, ZLINEAR, ZCUBIC, FALSE);

   while(!motion_done(0) || !motion_done(1) || !motion_done(2))
       ;
   error(clear_status(X));
   error(clear_status(Y));
   error(clear_status(Z));
}

int16 main()
{
   int16 error_code, axis, axes;
   double p;

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

   begin_jogging();
   axes = dsp_axes();
   printf("\nToggle bit #22 to latch %d axes.\n\n", axes);

   init_io(2, IO_INPUT);
   stop_with_latch();
   arm_latch(TRUE);

   while(!latch_status())
       ;
   for (axis = 0; axis < axes; axis++)
   {
       get_latched_position(axis, &p);
       printf("Axis:%d Position:%12.0lf\n", axis, p);
   }

   printf("\n");
   printf("Press any key to disable jogging and clear status on all axes...\n");
   getch();

   recover_from_latch();

   return 0;
}
```

Jog/latch positions, generate stop events, clear status on 3 axes

joglat2.c

Jog/latch positions, generate stop events & back off

Jogging

```
/* JOGLAT2.C

:Jog, latch positions, generate stop events, and back off a relative distance.

This code demonstrates how to configure the DSP for analog jogging, position latching, and Stop
Event generation based on an User I/O bit.  The steps are:

 1) Initialize the analog inputs and configure three axes for jogging
 2) Download frames to trigger Stop Events when User I/O bit #22 goes low
 3) Configure and arm position latching
 4) Wait for position latch and Stop Event
 5) Determine the last commanded direction
 6) Clear the Stop Events
 7) Command a relative move in the opposite direction

The jogging velocity is calculated as follows:

   Each sample, the DSP runs through the following calculations to
   determine the velocity that the motor will run at.

       if A/D value > (center + deadband)
           then J = A/D value - center - deadband
       if A/D value < (center - deadband)
           then J = A/D value - center + deadband
       if (center + deadband) > A/D value > (center - deadband)
           then J = 0

       motor velocity (counts/sec) =
        (linear term * 1/65536 * J + cubic term * 3.632E-12 * (J * J * J)) * sample_rate

Be sure user I/O bit #22 is normally driven high, and is pulled low to activate the latch.  The
falling edge of bit #22 triggers the DSP's interrupt.  The DSP's interrupt routine handles the
latching of the actual positions of all axes within 4 microseconds.

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
*/

/*  Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"
# include "idsp.h"


# define LATCH_BIT   22
# define VELOCITY    1000.0
```

```
# define ACCEL       10000.0

# define STOP_RATE   10000.0        /* Stop Event Deceleration */

# define EXP_EVENT   0x000D

int16
   jog_axes[] = {0, 1, 2},
   jog_channel[] = {0, 1, 2},
   jog_analog_channel[] = {0, 1, 2},
   jog_center[] = {2048, 2048, 2048},
   jog_deadband[] = {50, 50, 50},
   jog_linear[] = {5, 20, 50},
   jog_cubic[] = {10, 50, 100};

double back_off_dist[] = {7000.0, 8000.0, 5000.0};


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;
   }
}

void set_jogging(int16 n_axes, int16 * map, int16 enable)
{
   int16 i;

   for (i = 0; i < n_axes; i++)
       jog_axis(map[i], jog_channel[i], jog_center[i], jog_deadband[i],
            jog_linear[i], jog_cubic[i], enable);
}

void stop_when_latched(int16 n_axes, int16 * map)
{
   int16 i;

   /* Download frames to generate a Stop Event when the LATCH_BIT goes low. */

   for (i = 0; i < n_axes; i++)
   {
       dsp_io_trigger(map[i], LATCH_BIT, FALSE);
       dsp_axis_command(map[i], map[i], STOP_EVENT);
   }
}

void display(int16 n_axes, int16 * map)
{
    int16 i;
    double cmd;

    printf("\r");

    for (i = 0; i < n_axes; i++)
    {
       get_command(map[i], &cmd);
         printf("%d: %10.0lf ", i, cmd);
    }
```

```
}

int16 end_of_motion(int16 n_axes, int16 * map)
{
    int16 i;

    for(i = 0; i < n_axes; i++)
    {
       if(!motion_done(map[i]))
            return 0;
    }
    return 1;
}

void find_direction(int16 n_axes, int16 * map, int16 * direction)
{
   int16 i, temp_state, ts_addr;

   for (i = 0; i < n_axes; i++)
   {
       direction[i] = 0;

       ts_addr = dspPtr->pc_status + (dspPtr->axes * 2) + map[i];
       temp_state = dsp_read_dm(ts_addr);

       if ((temp_state & EXP_EVENT) == EXP_EVENT)
       {
           if (temp_state & TRIGGER_NEGATIVE)
              direction[i] = -1;
           else
              direction[i] = 1;
       }
   }
}

void recover(int16 n_axes, int16 * map)
{
   int16 i, dir[PCDSP_MAX_AXES], temp_state;

   set_jogging(n_axes, map, FALSE);         /* Disable jogging */

   while (!end_of_motion(n_axes, map))
       display(n_axes, jog_axes);

   find_direction(n_axes, map, dir);
   for (i = 0; i < n_axes; i++)
   {
       error(clear_status(map[i]));
       start_r_move(map[i], (dir[i] * back_off_dist[i]), VELOCITY, ACCEL);
   }
}

void initialize(int16 n_axes, int16 * map)
{
   int16 i, error_code;

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

   init_io(2, IO_INPUT);
   for (i = 0; i < n_axes; i++)
       set_stop_rate(map[i], STOP_RATE);

   /* Initialize the jog axes to use specific analog channels and configure
       these channels for unipolar and single ended operation.
       The default analog channel configuration is the analog channel = axis,
       unipolar, and single ended.
   */
```

```
   for (i = 0; i < n_axes; i++)
       set_analog_channel(map[i], jog_analog_channel[i], FALSE, FALSE);

   set_jogging(n_axes, jog_axes, TRUE);/* enable jogging */
   stop_when_latched(n_axes, jog_axes);
   arm_latch(TRUE);
}

int16 main()
{
   int16 i, n_axes;
   double position;

   n_axes = (sizeof(jog_axes)/sizeof(int16));

   initialize(n_axes, jog_axes);
   printf("\nToggle bit #22 to latch positions.\n\n");

   while (!latch_status())
       display(n_axes, jog_axes);

   for (i = 0; i < n_axes; i++)
   {
       get_latched_position(jog_axes[i], &position);
       printf("\nAxis: %d Latched Pos: %12.0lf", jog_axes[i], position);
   }
   printf("\n");

   recover(n_axes, jog_axes);
   while (!end_of_motion(n_axes, jog_axes))
       display(n_axes, jog_axes);

   return 0;
}
```

keylatch.c

Latch actual positions of all axes, using keyboard input

Position Latching

```
/* KEYLATCH.C

:Latch the actual positions of all axes based on keyboard input.

Be sure user I/O bit #22 is left unconnected when using this sample program. The falling edge of
bit #22 triggers the DSP's interrupt.  The DSP's interrupt routine handles the latching of the
actual positions of all axes within 4 microseconds.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"


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;
   }
}
```

```
int16 main()
{
   int16 error_code, axis, axes, done = 0;
   double p;

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

   axes = dsp_axes();
   printf("\nPress any key to latch %d axes, esc=quit\n", axes);

   init_io(2, IO_OUTPUT);
   set_bit(22);              /* initialize bit #22 high */
   arm_latch(TRUE);

   while (!done)
   {
       while (!latch_status())
       {
           if (kbhit())
           {
               if (getch() == 0x1B)
                   exit(1);
               else
                   latch();
           }
       }
       for (axis = 0; axis < axes; axis++)
       {   get_latched_position(axis, &p);
           printf("Axis:%d Position:%12.0lf\n", axis, p);
       }
       printf("\n");
       arm_latch(TRUE);      /* reset the latch. */
   }

    return 0;
}
```

Latch actual positions of all axes, using keyboard input

lcoord.c

Simple linear coordinated moves

Coordinated Motion

```
/* LCOORD.C

:Some simple linear coordinated moves

This code initializes the mapped axes, the vector velocity, and the vector acceleration.  Then a
single linear coordinated move is performed followed by an interpolated linear coordinated motion.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"


# define  MAX_AXES   3

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;
   }
}

int16 display(void)
{
   double x, y, z;

    while (!kbhit())
    {   get_command(0, &x);
        get_command(1, &y);
        get_command(2, &z);
        printf("X: %12.4lf Y: %12.4lf Z: %12.4lf\r", x, y, z);
    }
```

```
    getch();

    return 0;
}

int16 main()
{
   int16 error_code, axes[MAX_AXES] = {0, 1, 2};

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

    error(dsp_reset());             /* reset the hardware */

   error(map_axes(MAX_AXES, axes)); /* initialize the coordinated axes */
   set_move_speed(200.0);           /* set the vector velocity */
   set_move_accel(8000.0);          /* set the vector acceleration */

   move_3(100.0, 100.0, 100.0);     /* perform a single coordinated move */
    display();

   /*  Perform an interpolated motion through several points */
   start_point_list();
    move_3(100.0, 100.0, 100.0);
    move_3(200.0, 1000.0, 1000.0);
    move_3(50.0, 50.0, 50.0);
    end_point_list();

    start_motion();

    display();

    return 0;
}
```

limit.c

Recover from simple limit switch event

Exception Handling & Recovery


```
/* LIMIT.C

:Simple limit switch recovery

This sample demonstrates a basic limit switch recovery algorithm.  Each axis has a positive and
negative hardware limit input.  The active level and exception event response for each limit input
is software configurable.

Each limit input is logically combined with the command velocity.  The negative limit is disabled
when the axis moves in the positive direction and the positive limit is disabled when the axis
moves in the negative direction.

The DSP generates the exception event (None, Stop, E-Stop, or Abort) when the limit input reaches
the active state and the command velocity is non-zero in the direction of the corresponding limit
sensor.

Here is the algorithm:
 1) Velocity move towards the positive limit input
 2) Stop at the positive limit sensor (Stop Event generated by the DSP)
 3) Clear the Event
 4) Move off the positive limit sensor
 5) Velocity move towards the negative limit input
 6) E-Stop at the negative limit sensor (E-Stop Event generated by the DSP)
 7) Clear the Event
 8) Move off the negative limit sensor

Here is the sensor logic:
 + limit input = active low
 - limit input = active low

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
*/

/*  Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/

# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"

# define AXIS               0
# define VEL                1000.0
# define ACCEL              5000.0
# define RECOVER_DIST       4000.0
```

```
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;
   }
}

void display(int16 axis)
{
   int16 pos_lim, neg_lim, state;
   double cmd;

   while (!motion_done(axis))
   {   get_command(axis, &cmd);
       pos_lim = pos_switch(AXIS);
       neg_lim = neg_switch(AXIS);
       printf("Cmd:%8.0lf +Lim:%d -Lim:%d\r", cmd, pos_lim, neg_lim);

       if (kbhit())
       {   getch();
           v_move(axis, 0.0, ACCEL);
           exit(1);
       }
   }
   printf("\n");
}

int16 check_for_event(int16 axis)
{
   int16 state;

   state = axis_state(axis);

   /* Check if an Exception Event occurred */
   switch (state)
   {
       case STOP_EVENT:
       {   printf("\nException Event = Stop");
           return state;
       }

       case E_STOP_EVENT:
       {   printf("\nException Event = Emergency Stop");
           return state;
       }
   }
   return 0;
}

int16 recover(int16 axis, int16 state)
{
   int16 recover_dir;

   switch (axis_source(axis))
   {
       case ID_POS_LIMIT:
       {   printf("\nSource = Positive Limit");
           recover_dir = -1;
           break;
```

limit.c

Recover from simple limit switch event

```
        }

        case ID_NEG_LIMIT:
        {   printf("\nSource = Negative Limit");
            recover_dir = 1;
            break;
        }

        default:
            printf("\nNo Limit was triggered\n");
    }

    /* Make sure the event is complete before clearing the status */
    while (!motion_done(axis))
        ;

    if ((state == E_STOP_EVENT) || (state == STOP_EVENT))
    {   error(clear_status(axis));

        printf("\nMoving off the limit\n");
        start_r_move(axis, (RECOVER_DIST * recover_dir), VEL, ACCEL);

        display(axis);
    }

    /* Was the recovery successful? */
    if (check_for_event(axis))
    {   printf("\nUnable to Recover - Both limits tripped?\n");
        return 1;
    }

    return 0;
}

int16 main()
{
    int16 error_code, state;

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

    /* Configure the limits */
    set_positive_limit(AXIS, STOP_EVENT);
    set_negative_limit(AXIS, E_STOP_EVENT);
    set_positive_level(AXIS, FALSE);
    set_negative_level(AXIS, FALSE);
    set_stop_rate(AXIS, ACCEL);
    set_e_stop_rate(AXIS, ACCEL);

    printf("\nSearching for the positive limit sensor. (any key to quit)\n\n");
    v_move(AXIS, VEL, ACCEL);        /* velocity move towards positive limit */
    display(AXIS);

    state = check_for_event(AXIS);   /* Did we hit the positive limit? */
    if (state)
        recover(AXIS, state);

    printf("\nSearching for the negative limit sensor. (any key to quit)\n\n");
    v_move(AXIS, -VEL, ACCEL);       /* velocity move towards positive limit */
    display(AXIS);

    state = check_for_event(AXIS);   /* Did we hit the negative limit? */
    if (state)
        recover(AXIS, state);

    return 0;
}
```

linksync.c

Electronic gearing with synch, using I/O sensors

Electronic Gearing

```
/* LINKSYNC.C

:Electronic gearing with synchronization based on I/O sensors.

This sample code link's 2 axes together and commands a velocity move on the master axis.  During
the motion, the actual position of the slave axis is displayed based on 2 input sensors.  For this
sample, the index inputs are used as the input sensors.

Based on keyboard input, a synchronization routine adjusts the offset distance between the input
sensors.

Here is the synchronization algorithm:

 1) Read the slave's position when the first sensor goes high
 2) Read the slave's position when the second sensor goes high
 3) Calculate the offset distance
 4) Command a motion to compensate for the offset distance

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include <math.h>
# include "pcdsp.h"

# define MASTER      0
# define SLAVE       1
# define VELOCITY    500.0
# define ACCEL       4000.0

# define RATIO       1.0
# define MAX_DIST    8192    /* Slave distance between sensors */

# define SENSOR_PORT 6       /* User I/O port */
# define M_MASK      0x04    /* Master sensor mask */
# define S_MASK      0x40    /* Slave sensor mask */
```

```
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;
   }
}

void synchronize(int16 axis, int16 port, int16 m_mask, int16 s_mask)
{
   int16 dir, value = 0;
   double delta, offset, s_pos_0, s_pos_1;

   while (!(value & m_mask))        /* wait for master sensor */
       get_io(port, &value);
   get_position(axis, &s_pos_0);

   while (!(value & s_mask))        /* wait for slave sensor */
       get_io(port, &value);
   get_position(axis, &s_pos_1);

   delta = s_pos_1 - s_pos_0;

   if (delta > 0.0)
       dir = 1;
   else
       dir = -1;

   printf("\nDir: %d Delta: %6.0lf", dir, delta);
   /* Make sure the delta is a fraction of one full sensor cycle. */
   delta = fmod((dir * delta), MAX_DIST);

   /* Calculate the shortest distance to synchronize the master and slave. */
   if (delta > (MAX_DIST / 2.0))
       offset = (dir * delta) - (dir * MAX_DIST);
   else
       offset = (dir * delta);

   /* Command a relative move to compensate for the offset distance. */
   start_r_move(axis, offset, VELOCITY, ACCEL);
   printf("\nOffset: %6.0lf\n\n", offset);
}

void display(int16 axis, int16 sensor_port, int16 m_mask, int16 s_mask)
{
   int16 value;

   get_io(sensor_port, &value);

   if ((value & m_mask) || (value & s_mask))
       printf("I/O: 0x0%x Enc: %d\n", value, dsp_encoder(axis));
}
```

```
int16 main()
{
   int16 error_code, key, done = 0;

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

   set_home_index_config(MASTER, INDEX_ONLY);      /* use index for simulation */
   set_home_index_config(SLAVE, INDEX_ONLY);

   mei_link(MASTER, SLAVE, RATIO, LINK_ACTUAL);
   v_move(MASTER, VELOCITY, ACCEL);         /* simulate master */

   printf("\ns=synchronize, esc=quit\n");

   while (!done)
   {   display(SLAVE, SENSOR_PORT, M_MASK, S_MASK);

       if (kbhit())/* key pressed? */
       {   key = getch();

           switch (key)
           {
               case 's':/* Synchronize */
                   printf("\nSynchronizing...\n");
                   synchronize(SLAVE, SENSOR_PORT, M_MASK, S_MASK);
                   break;

               case 0x1B:/* <ESC> */
                   v_move(MASTER, 0.0, ACCEL);       /* Stop the master */
                   while (!motion_done(MASTER))
                       ;
                   endlink(SLAVE);
                   done = TRUE;
                   break;
           }
       }
   }
   return 0;
}
```

lsint.c

Link synch using I/O sensors in interrupt routine

Electronic Gearing

```
/* LSINT.C

:Link synchronization based on I/O sensors in an interrupt routine.

This sample link's 2 axes together and commands a velocity move on the master axis.  Then the DSP
is configured to interrupt the host CPU every sample.  The "delta" distance and current link
"ratio" are displayed.  The axes are synchronized based on keyboard input.

The interrupt routine is as follows:

 1) Read the sensor inputs (for simulation purposes, read the index inputs)
 2) Read the 16 bit encoder inputs and update the 32 bit software position
 3) Determine the positions at the sensors
 4) If "sync_flag" is TRUE, synchronize the axes

Here is the synchronization algorithm:

 1) Determine the slave positions at the sensors
 2) Check if the sync_flag is enabled
 3) Check if the sync timer has expired
 4) Calculate the delta distance between sensor positions
 5) If the delta has changed and is larger than the deadband, then calculate
      a new link ratio:
           ratio = initial_ratio + (delta * P_RATIO);
 6) Convert the new link ratio to 16 bit whole, 16 bit fractional
 7) Download a frame to set the new link ratio

The link ratio is calculated based on the slave's "delta" distance between the master sensor and
the slave sensor.  Then the delta is multiplied by the proportional term, P_RATIO.

The link ratio is represented as a 32 bit signed value.  The upper 16 bits represent the whole
portion and the lower 16 bits represent the fractional portion.

There are several parameters to control the response of the synchronization algorithm that depend
on the mechanical system:

   RATIO -        Initial link ratio (floating point)
   MIN_RATIO -    Minimum link ratio boundary (16 bit whole, 16 bit fractional)
   MAX_RATIO -    Maximum link ratio boundary (16 bit whole, 16 bit fractional)
   P_RATIO -      Proportional term for synchronization response,
                   (16 bit whole, 16 bit fractional)
   SYNC_DELAY -   Number of samples to delay before next link ratio update
   MAX_DIST -     Distance between sensors on slave axis
   DEADBAND -     Maximum tolerance between master and slave synchronization


When programming with interrupt routines, make sure to define CRITICAL and ENDCRITICAL.  These are
located in idsp.h.  CRITICAL is used to disable interrupts and ENDCRITICAL re-enables interrupts.
This is very important! Interrupts during communication can cause strange problems.

If you are using a Borland C compiler (3.1 or greater), then CRITICAL and ENDCRITICAL are defined
for you.

When compiling code with an interrupt routine, turn stack checking off.

If you modify dsp_interrupt(...), watch out for floating point operations, most compilers do not
support them.  Also, MEI's standard function library is not supported inside interrupt routines.

Written for Borland and Microsoft compilers.
```

```
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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <dos.h>
# include <conio.h>
# include "idsp.h"


# define R_8259_EOI         0x20
# define R_8259_IMR         0x21
# define EOI                0x20

# define MASTER      0
# define SLAVE       1
# define VELOCITY    700.0
# define ACCEL       4000.0

# define RATIO       1.0           /* used to set the initial link */
# define MIN_RATIO   0.0
# define MAX_RATIO   2.0
# define P_RATIO     8             /* proportional term for synchronization,
                                        16 bit whole, 16 bit fractional */
# define SYNC_DELAY  20            /* number of samples to delay next update */

# define MAX_DIST           8192          /* Slave distance between sensors */
# define HALF_MAX_DIST      (MAX_DIST / 2)
# define DEADBAND           100           /* Maximum tolerance */

# define MAX_IO_PORTS       9
# define SENSOR_PORT        6             /* I/O port with home/index inputs */
# define SENSOR_ADDR        0x30          /* Address of home input port */
# define M_MASK             0x04          /* Master sensor mask */
# define S_MASK             0x40          /* Slave sensor mask */


int16
   encoder[PCDSP_MAX_AXES],         /* encoder counts */
   act_velocity[PCDSP_MAX_AXES],    /* counts per sample */
   io_port[MAX_IO_PORTS],
   sync_flag = 0,
   whole_ratio,
   dir;

unsigned16
   frac_ratio,
   sync_cnt = SYNC_DELAY;

int32
   act_position[PCDSP_MAX_AXES],
   s_pos_0,
   s_pos_1,
   initial_ratio,
   ratio,
   min_ratio,
```

lsint.c

```
   max_ratio,
   delta;

typedef void (interrupt * INTERRUPT_HANDLER)();
INTERRUPT_HANDLER old_handler = NULL;
static int16 imr, interrupt_number = 5;/* IRQ 5 */


static int16 Read_DSP(unsigned addr)
{
   outpw(dspPtr->address,addr | 0x8000);
   return(inpw(dspPtr->data));
}

static void Write_DSP(unsigned addr, int16 dm)
{
   outpw(dspPtr->address,addr | 0x8000);
   outpw(dspPtr->data,dm);
}

static int16 set_dsp_irq(int16 axis, int16 enable)
{
    int16 addr1,addr2;
    int16 v1, v2 ;

    addr1 = dspPtr->global_data + axis + GD_SIZE;          /* irq_count */
    addr2 = addr1 + dspPtr->axes;                          /* ack_count */
    v1 = Read_DSP(addr1);
    v2 = Read_DSP(addr2);

    if (enable)
      Write_DSP(addr2, v1 + 1);     /* DSP generate interrupts every sample */
    else
      Write_DSP(addr2, v1);         /* set ack_count = irq_count */

    return (v1 - v2);
}

void link_frame(int16 slave_axis, int16 whole, unsigned16 frac)
{
   FRAME frame;
   DSP_DM addr = dspPtr->data_struct + DS(slave_axis) + DS_RATIO, buffer[2];

   buffer[0] = frac;
   buffer[1] = whole;

   dsp_move_frame(&frame, slave_axis, MF_MOVE, addr, MF_DATA_AREA,
       MF_DATA_AREA, 2, buffer);
}
```

Link synch using I/O sensors in interrupt routine

```
void _far interrupt dsp_interrupt()
{
   int16 axis, new_enc;
   int32 new_delta;

   disable();

   io_port[SENSOR_PORT] = Read_DSP(SENSOR_ADDR);    /* read the index inputs */

   for (axis = 0; axis < PCDSP_MAX_AXES; axis++)
   {   new_enc = Read_DSP(ENCODER_0 + axis);
       act_velocity[axis] = new_enc - encoder[axis];
       act_position[axis] += act_velocity[axis];
       encoder[axis] = new_enc;
   }

   if (io_port[SENSOR_PORT] & M_MASK)               /* Master index pulse */
       s_pos_0 = act_position[SLAVE];
   if (io_port[SENSOR_PORT] & S_MASK)               /* Slave index pulse */
       s_pos_1 = act_position[SLAVE];

   if (sync_flag)
   {
       if (!sync_cnt)
       {
           new_delta = s_pos_1 - s_pos_0;
           new_delta %= MAX_DIST;

           if (new_delta != delta)
           {
               delta = new_delta;

               if (delta > 0)
                   dir = 1;
               else
                   dir = -1;

               if ((delta * dir) > DEADBAND)
               {
                   if ((dir * delta) < HALF_MAX_DIST)
                       ratio = initial_ratio + (delta * P_RATIO);
                   else
                       ratio = initial_ratio - (delta * P_RATIO);
               }
               else
                   ratio = initial_ratio;

               if (ratio > max_ratio)
                   ratio = max_ratio;
               if (ratio < min_ratio)
                   ratio = min_ratio;

               whole_ratio = (int16)(ratio >> 16);
               frac_ratio = (unsigned16)(ratio & 0xFFFF);
               link_frame(SLAVE, whole_ratio, frac_ratio);
               sync_cnt = SYNC_DELAY; /* reset sync count down */
           }
       }
       else
           sync_cnt--;
   }
    outp(R_8259_EOI, EOI);
}
```

```
void remove_handler(void)
{
   _disable() ;
   _dos_setvect(interrupt_number + 8, old_handler) ;
   outp(R_8259_IMR, imr) ;
   outp(R_8259_EOI, EOI) ;
   _enable() ;
}

void install_dsp_handler(int16 intno)
{
   int16 new_imr;

   interrupt_number = intno ;
   old_handler = _dos_getvect(interrupt_number + 8) ;
   imr = inp(R_8259_IMR) ;

   atexit(remove_handler) ;

   new_imr = imr & ~(1 << interrupt_number) ;
   _disable() ;
   _dos_setvect(interrupt_number + 8, dsp_interrupt) ;
   outp(R_8259_IMR, new_imr) ;
   outp(R_8259_EOI, EOI) ;
   _enable() ;
}

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;
   }
}

void initialize(void)
{
   int16 axis, error_code;
   double x, src, w, f;

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

   /* This is really only needed for code that doesn't do a dsp_reset. */
   for (axis = 0; axis < PCDSP_MAX_AXES; axis++)
   {   get_position(axis, &x);
       act_position[axis] = (long)x;
       encoder[axis] = dsp_encoder(axis);
   }

   set_home_index_config(MASTER, INDEX_ONLY);      /* use index for simulation */
   set_home_index_config(SLAVE, INDEX_ONLY);

   mei_link(MASTER, SLAVE, RATIO, LINK_ACTUAL);
   v_move(MASTER, VELOCITY, ACCEL);  /* simulate master */
```

```
   /* Convert link ratio from floating point to fixed point format,
       16 bit signed whole, 16 bit unsigned fractional */
   initial_ratio = ratio = (int32) (RATIO * SFRACTIONS_PER_COUNT);
   min_ratio = (int32) (MIN_RATIO * SFRACTIONS_PER_COUNT);
   max_ratio = (int32) (MAX_RATIO * SFRACTIONS_PER_COUNT);
   printf("\nInitial Ratio: %ld\n", initial_ratio);

   install_dsp_handler(interrupt_number);
   init_io(2, IO_OUTPUT);           /* PC interrupt bit is located on port 2 */
   reset_bit(23);                   /* enable interrupts */
   set_dsp_irq(0, TRUE);            /* interrupt host CPU every sample */
}

int16 main()
{
   int16 key, done = 0;

   initialize();
   printf("\ns=turn synchronize on/off, esc=quit\n");

   while (!done)
   {
       printf("Delta:%8ld Sync:%1d Whole:0x%4x Frac:0x%4x\r",
            delta, sync_flag, whole_ratio, frac_ratio);

       if (kbhit())/* key pressed? */
       {   key = getch();

           switch (key)
           {
               case 's':                            /* Turn on/off synchronization */
                   if (sync_flag)
                       sync_flag = 0;
                   else
                       sync_flag = 1;
                   break;

               case 0x1B:/* <ESC> */
                   v_move(MASTER, 0.0, ACCEL);      /* Stop the master */
                   done = TRUE;
                   break;
           }
       }
   }
   while (!motion_done(MASTER))
       ;
   endlink(SLAVE);
   set_bit(23);                                     /* disable interrupts */
   set_dsp_irq(0, FALSE);                           /* turn off interrupts */

   return 0;
}
```

masync.c

Multi-axis synchronized motion

Synchronized Motion

```
/* MASYNC.C

:Multiple axis synchronized motion.

This code demonstrates how to generate sequences of multiple axis S-Curve profile motion.  The
execution of the profiles are synchronized so that all of the axes start in the same sample.

The steps are:
 1) Set the DSP's gate flag for each axis
 2) Download the profiles.
    Be sure to turn on the Hold flag in the first frame of each sequence.
    Also, load a dwell frame and an end sequence frame on axes that are not commanded to move
 3) Reset the DSP's gate flag for each axis.
    The frames will execute.
 4) Wait for the dwell frame and end sequence frames to execute (about 2 samples)
 5) Download the next frame sequence for each axis
 6) Wait for the previous motion to complete
 7) Reset the DSP's gate flag for each axis.
    The next frame sequences will execute.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include "pcdsp.h"
# include "idsp.h"


double
   pos[] =    {50000.0, 60000.0, 85000.0, 90000},
   vel[] =    {4000.0, 4000.0, 4000.0, 4000.0},
   accel[] =  {16000.0, 16000.0, 16000.0, 16000.0},
   jerk[] =   {100000.0, 100000.0, 100000.0, 100000.0};
```

```
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;
   }
}

void display(int16 n_axes, int16 * map)
{
    int16 i;
    double cmd;

    for (i = 0; i < n_axes; i++)
    {
       get_command(map[i], &cmd);
         printf("%d: %10.0lf ", map[i], cmd);
    }
    printf("\r");
}

int16 seq_done(int16 n_axes, int16 mask, int16 * map)
{
   int16 i, axis;

   /* Check if the frame sequence on the non-motion axes is completed.
       This check is required before calling set_gates(...) to guarantee
       the synchronization of the axes.
   */

   for (i = 0; i < n_axes; i++)
   {
       axis = map[i];
       if ((~mask & (1 << axis)) && (!motion_done(axis)))
           return 0;
   }
   return 1;
}

int16 moving(int16 n_axes, int16 * map)
{
   int16 i;

   for (i = 0; i < n_axes; i++)
   {
       if (in_motion(map[i]))
           return 1;
   }
   return 0;
}
```

masync.c

```
int16 multi_s_move(int16 n_axes, int16 mask, int16 * map)
{
   int16 axis, i;

   set_gates(n_axes, map);                 /* prevent frames from executing */

   for (i = 0; i < n_axes; i++)
   {
       axis = map[i];
       if (mask & (1 << axis))
           start_s_move(axis, pos[i], vel[i], accel[i], jerk[i]);
       else
       {   /* Turn on the Hold bit in the first frame */
           dsp_control(axis, FCTL_HOLD, TRUE);
           dsp_dwell(axis, 0.0);     /* one sample delay */
           dsp_control(axis, FCTL_HOLD, FALSE);
           dsp_end_sequence(axis);
       }
   }

   /* Wait for the previous motion to complete before starting this one */
   while (moving(n_axes, map))
       display(n_axes, map);

   reset_gates(n_axes, map);/* ready, set, go! */

   return 0;
}

int16 main()
{
   int16 error_code, n_axes, axis_mask, axes[] = {0, 1, 2, 3};

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

   n_axes = (sizeof(axes) / sizeof(int16));

   axis_mask = 3;     /* axes 0 and 1 */
   multi_s_move(n_axes, axis_mask, axes);
   while (!seq_done(n_axes, axis_mask, axes))
       ;

   axis_mask = 1;     /* axis 0 */
   pos[0] = 110000.0;
   multi_s_move(n_axes, axis_mask, axes);
   while (!seq_done(n_axes, axis_mask, axes))
       ;

   axis_mask = 3;     /* axes 0 and 1 */
   pos[0] = 0.0;
   pos[1] = 0.0;
   multi_s_move(n_axes, axis_mask, axes);
   while (!seq_done(n_axes, axis_mask, axes))
       ;
   while (moving(n_axes, axes))
       display(n_axes, axes);

    return 0;
}
```

minthnd.c

Multi-board interrupt support

Interrupt Handling

```
/* MINTHND.C

:Sample code for Multi-Board Interrupt Support under DOS.

This code demonstrates how to handle interrupts generated from multiple DSP-Series controllers.
Sharing interrupts requires an external pull-up resistor on the IRQ line.  The latest revisions of
the surface mount controllers have this resistor.  The PC/DSP does not have this resistor.

When programming with interrupt routines, make sure to define CRITICAL and ENDCRITICAL.  These are
located in idsp.h.  CRITICAL is used to disable interrupts and ENDCRITICAL re-enables interrupts.
This is very important! Interrupts during communication can cause strange problems.

If you are using a Borland C compiler (3.1 or greater), then CRITICAL and ENDCRITICAL are defined
for you.

When compiling code with an interrupt routine, turn stack checking off.

If you modify dsp_interrupt(...), watch out for floating point operations, most compilers do not
support them.  Also, MEI's standard function library is not supported inside interrupt routines.

Written for Borland and Microsoft compilers.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <dos.h>
# include <conio.h>
# include "idsp.h"
# include "mboard.h"

# define  R_8259_EOI        0x20
# define  R_8259_IMR        0x21
# define  EOI               0x20
# define  EF_NONE           (-1)
# define  DATA_STRUCT(dsp, axis, offset)((dsp)->data_struct + (DS_SIZE * (axis)) + offset)

typedef void (interrupt * INTERRUPT_HANDLER)();

int16 set_board(int16 boardno);
static int16 Read_DSP(unsigned addr);
static void Write_DSP(unsigned addr, int16 dm);
void install_dsp_handler(int16 intno);
typedef void (interrupt * INTERRUPT_HANDLER)();
INTERRUPT_HANDLER old_handler = NULL;
```

```
static int16 imr, interrupt_number;

#  define BOARDS  3

int16 board = 0;
extern int16 Boards;

typedef int16 I_LIST [PCDSP_MAX_AXES];

I_LIST
   axis_interrupt_list[BOARDS];

int16 * axis_interrupts;
int16 io_interrupts[BOARDS];

static int16 inc_acks(int16 axis)
{
    int16 addr1,addr2;
    int16 v1, v2 ;

    addr1 = dspPtr->global_data + axis + GD_SIZE;           /* irq_count */
    addr2 = addr1 + dspPtr->axes;                           /* ack_count */
    v1 = Read_DSP(addr1);
    v2 = Read_DSP(addr2);

    if (v1 != v2)                                  /* Is the DSP generating interrupts? */
      Write_DSP(addr2,v1);/* set ack_count = irq_count */
    return (v1 - v2);
}

void _far interrupt dsp_interrupt()
{
   int16 i, l, m;

   _disable();

   m = board;

   for (l = 0; l < Boards; l++)
   {
       set_board(l);

       /* Is the interrupt caused by the DSP I/O monitoring? */
       if (Read_DSP((Read_DSP(DM_IO_BLOCK)) + IO_CHANGE))
       {
           io_interrupts[l]++;
           /* Clear the I/O monitoring flag. */
           Write_DSP((Read_DSP(DM_IO_BLOCK) + IO_CHANGE), 0);
       }

       for (i = 0; i < PCDSP_MAX_AXES; i++)
       {
           if (inc_acks(i))
           {
              axis_interrupts[i]++;
           }
       }
   }

   set_board(m);
    outp(R_8259_EOI, EOI);
}
```

```
void remove_handler(void)
{
   _disable() ;
   _dos_setvect(interrupt_number + 8, old_handler) ;
   outp(R_8259_IMR, imr) ;
   outp(R_8259_EOI, EOI) ;
   _enable() ;

}

void install_dsp_handler(int16 intno)
{
   int16 new_imr;

   interrupt_number = intno ;
   old_handler = _dos_getvect(interrupt_number + 8) ;
   imr = inp(R_8259_IMR) ;

   atexit(remove_handler) ;

   new_imr = imr & ~(1 << interrupt_number) ;
   _disable() ;
   _dos_setvect(interrupt_number + 8, dsp_interrupt) ;
   outp(R_8259_IMR, new_imr) ;
   outp(R_8259_EOI, EOI) ;
   _enable() ;

}

int16 dsp_irq_frame(int16 axis)
{
   FRAME frame ;
   frame_clear(&frame)  ;
   if (frame_allocate(&frame, dspPtr, axis))
       return dsp_error ;
   frame.f.control |= FCTL_INTERRUPT ;
   return frame_download(&frame) ;
}

int16 set_board(int16 boardno)
{
   board = boardno;
   axis_interrupts = axis_interrupt_list[board];
   return m_board(boardno);
}

static int16 Read_DSP(unsigned addr)
{  outpw(dspPtr->address,addr | 0x8000);
   return(inpw(dspPtr->data));
}

static void Write_DSP(unsigned addr, int16 dm)
{
   outpw(dspPtr->address,addr | 0x8000);
   outpw(dspPtr->data,dm);
}
```

mtask.c

Multi-tasking under WinNT

Windows NT

```
/* MTASK.C

:Multi-tasking under Windows NT.

This sample demonstrates how to create multiple threads under Windows NT.

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 <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "idsp.h"

#ifdef MEI_MSVC20           /* support for Microsoft Visual C/C++ ver 2.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif

#ifdef MEI_MSVC40           /* support for Microsoft Visual C/C++ ver 4.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif

#define   SEM_MAXIMUM_USAGE               1L
#define   SEM_INITIAL_SIGNALED            1L
#define   SEM_INITIAL_NON_SIGNALED        0L
#define   SEM_INCREMENT                   1L
#define   THREAD_STACK_SIZE               0
#define   THREAD_CREATION_FLAGS           0
#define   EXIT_CODE_OK                    0

/* Globals */
HANDLE        MEI_SEMAPHORE;        /* Handle to library semaphore */
HANDLE        hTasks[2];            /* Handles to each created task */
int16         port = 0, io;


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;
   }
}
```

```
/* Create the semaphore object.  This semaphore will allow access to the
   MEI library for one task at time.  All other tasks will wait on the
   semaphore until it is available.  The initial state of the semaphore
   is signaled. */
HANDLE create_MEI_semaphore(void)
{  return CreateSemaphore(NULL, SEM_INITIAL_SIGNALED,
       SEM_MAXIMUM_USAGE, "MEI_semaphore");
}

/* Terminate the semaphore object */
int16 kill_MEI_semaphore(HANDLE sem)
{  return (int16)CloseHandle(sem);
}

/* Wait for the semaphore to be signaled and set the semaphore to a
   non-signaled state */
int16 get_MEI_semaphore(HANDLE sem)
{  return (int16)WaitForSingleObject(sem, INFINITE);
}

/* Release the semaphore by incrementing it to a signaled state */
int16 release_MEI_semaphore(HANDLE sem)
{  return ReleaseSemaphore(sem, SEM_INCREMENT, NULL);
}

void MEI_motion_task(void *args)
{  int16 axis = 0;
   while(1)
   {   get_MEI_semaphore(MEI_SEMAPHORE);
           if(motion_done(axis))
               start_move(axis, 10000.0, 10000.0, 100000.0);
           if(motion_done(axis))
               start_move(axis, 0.0, 10000.0, 100000.0);
       release_MEI_semaphore(MEI_SEMAPHORE);
   }
}

void MEI_io_task(void *args)
{
   while(1)
   {   get_MEI_semaphore(MEI_SEMAPHORE);
           get_io(port, &io);
       release_MEI_semaphore(MEI_SEMAPHORE);
   }
}
```

mtask.c

```
int main()
{  unsigned long ThreadId[2], i;
   int16 error_code;

   MEI_SEMAPHORE = create_MEI_semaphore();

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

   init_io(PORT_A, IO_INPUT);       /* initialize I/O port for inputs */

   /* create thread
       default security descriptor, default stack size (same as creating thread)*/
   hTasks[0] = CreateThread(NULL, THREAD_STACK_SIZE,
       (LPTHREAD_START_ROUTINE)MEI_motion_task, NULL,
       THREAD_CREATION_FLAGS, &ThreadId[0]);

   hTasks[1] = CreateThread(NULL, THREAD_STACK_SIZE,
       (LPTHREAD_START_ROUTINE)MEI_io_task, NULL,
       THREAD_CREATION_FLAGS, &ThreadId[1]);

   while(!kbhit())
       printf("Port %d\t value: %4d\r", port, io);
   getch();
   for(i = 0; i < 2; i++)
   {   TerminateThread(hTasks[i], EXIT_CODE_OK);
       CloseHandle(hTasks[i]);
   }
   kill_MEI_semaphore(MEI_SEMAPHORE);
   return 0;
}
```

oscdata1.c

SERCOS: Oscilloscope data acquisition with Indramat drive

SERCOS

```
/* OSCDATA1.C

:Oscilloscope data acquisition with a SERCOS Indramat drive.

This sample demonstrates how to configure a SERCOS Indramat drive to acquire data samples with its
oscilloscope functions.  The Indramat drive can be configured to capture a specified buffer of
data.  Data can be position, velocity, or torque information.  Data collection can be triggered
externally or internally.  Also, trigger offsets and delays can be specified.  For more informa-
tion regarding the oscilloscope functions, please consult the Indramat drive documentation.

Be sure to initialize the SERCOS ring with serc_reset(...) before reading/writing to the IDNs.

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 Motion Library Version 2.5
*/


#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "pcdsp.h"
#include "idsp.h"
#include "sercos.h"

#ifdef MEI_MSVC20           /* support for Microsoft Visual C/C++ ver 2.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif

#ifdef MEI_MSVC40           /* support for Microsoft Visual C/C++ ver 4.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif

#define PIDN 32768          /* drive specific IDNs */

unsigned16 data[2048];


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;
   }
}
```

oscdata1.c

```
int main()
{
   int16 i, error_code, n_words;
   long value;
   unsigned16 channel;

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

   /* Configure Oscilloscope data logging:
       0 : no signal feedback
       1 : position feedback
       2 : velocity feedback
       3 : velocity deviation
       4 : position deviation
       5 : torque command value
   */
   error(set_idn(0, 23 + PIDN, 1));

   error(set_idn(0, 25 + PIDN, 2));         /* use internal triggering */
   error(set_idn(0, 26 + PIDN, 1));         /* trigger with position feedback */
   error(set_idn(0, 27 + PIDN, 0L));        /* trigger at position zero */
   error(set_idn(0, 30 + PIDN, 3));         /* trigger in pos/neg direction */

   error(set_idn(0, 31 + PIDN, 250));       /* acquire at 250 microsec intervals */

   error(set_idn(0, 32 + PIDN, 512));       /* memory allocation */
   error(set_idn(0, 33 + PIDN, 2));         /* trigger sample delay */

   error(set_idn(0, 36 + PIDN, 0x7));       /* start */

   while (!kbhit())
   {   error(get_idn(0, 37 + PIDN, &value));       /* read status */
       printf("\rValue:0x%x", value);
   }
   getch();

   channel=dspPtr->sercdata[0].channel;
   read_idn(channel, 21 + PIDN, &n_words, data, TRUE); /* read data */

   printf("\nn_words:%d", n_words);
   for (i = 0; i < n_words; i++)
       printf("\nSample:%4d Value:%u", i, data[i]);

    return 0;
}
```

p3cfg.c

Reconfigure Dedicated I/O (axes 4-7) as User I/O

I/O Configurations

```
/* P3CFG.C

:Reconfigure Dedicated I/O (axes 4-7) as User I/O

This sample demonstrates how to reconfigure the dedicated I/O for axes 4-7 as User I/O.  The DSP
configures the I/O based on the number of firmware axes. Each axis uses 2 outputs and 4 inputs as
dedicated I/O.  Normally, Header P2 is used as dedicated I/O for axes 0-3 and P3 is used as dedi-
cated I/O for axes 4-7.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include "pcdsp.h"
# include "idsp.h"


void error(int 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;
   }
}
```

p3cfg.c

```
int16 main()
{
    int16 error_code, axis, addr;

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

  /*  Prevent DSP from accessing Dedicated I/O (axes 4-7) */
  for (axis = 4; axis < 8; axis++)
  {
      addr = dspPtr->e_data+ED(axis)+ED_STATUS_PORT;
      dsp_write_dm(addr, 0x95);                    /* redirect DSP to unused memory */
      set_index_required(axis, FALSE);             /* disable home and index logic */
  }

    return 0;
}
```

Reconfigure Dedicated I/O (axes 4-7) as User I/O

pidchng.c

Change PID filter params when error limit is exceeded

PID Filter

```
/* PIDCHNG.C

:Change the PID filter parameters when the error limit is exceeded.

This sample demonstrates how to use a frame to update the PID filter parameters when an axis'
error limit is exceeded.  The error limit is configured to generate a New Frame Event.  Then the
frame buffer is loaded with a very long dwell frame and a dsp_set_filter(...) frame.  When the
error limit is exceeded, the DSP will throw out the dwell frame and execute the
dsp_set_filter(...) frame.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"

# define AXIS    0


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;
   }
}
```

pidchng.c

Change PID filter params when error limit is exceeded

```
void display(int16 axis)
{
   int16 coeffs[COEFFICIENTS];
   double error;

   while (!kbhit())
   {   get_error(axis, &error);
       get_filter(axis, coeffs);
       printf("Error: %6.0lf P: %6d\r", error, coeffs[0]);
   }
   getch();
}

int16 main()
{
   int16
       error_code,
       pid_coeffs[COEFFICIENTS],
       soft_coeffs[COEFFICIENTS] = {80, 7, 300, 0, 0, 32767, 29, 32767, -9};

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

   get_filter(AXIS, pid_coeffs);
   set_error_limit(AXIS, 300.0, NEW_FRAME);

   dsp_dwell(AXIS, 1000000.0);              /* DSP will wait for error limit */
    dsp_set_filter(AXIS, soft_coeffs);      /* DSP will change filter params */

    display(AXIS);

    return 0;
}
```

pvt4.c

Generate multi-axis coord motion profile using frames

Host Motion Profile Trajectory Calculation

```
/* PVT4.C

:Generate a multi-axis coordinated motion profile using frames.

This sample shows how to update jerk, acceleration, velocity, and position at specified times
using frames.

Each frame is a 20 word structure that contains information about the motion trajectory.  Every
sample, the DSP's calculates an axis' command position based on the jerk, acceleration, and veloc-
ity values.  The frames are used to load the values for the jerk, acceleration, velocity, and/or
command position.

The frames are stored in an on-board buffer.  The buffer can hold a maximum of 600 frames.  The
DSP executes each frame and then releases it back to the "free list".

Here are the steps to generate a multi-axis coordinated motion profile.  The downloading of the
frames is based on a variable called frame_index:

frame_index == FIRST_FRAME
 1) Set the gate flag for each axis to prevent the frames from executing.
 2) Download the first frame for each axis.  Be sure to set the control word
   to enable the hold bit and check frames.  Also, set the action for check
   frames.  The check frames will cause the DSP to generate an E-Stop if a
   valid next frame does not exist.
 3) Reset the gate flag(s) to start the frame execution.

frame_index == MID_FRAME
 1) Download the next frame in the list.  Be sure to set the control word to
   enable check frames and set the action for check frames.

frame_index == LAST_FRAME
 1) Download the last frame.  Be sure to set the action word to 0 (to clear the
   in_sequence flag).

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <conio.h>
# include <dos.h>
# include <stdlib.h>
# include <math.h>
# include "pcdsp.h"
# include "idsp.h"
```

pvt4.c

Generate multi-axis coord motion profile using frames

```
# define BUFFER_SIZE        300           /* number of frames */
# define MAX_POINTS         4

# define ACCEL    1000.0                  /* counts per sec * sec */
# define TIME     10.0                    /* time between frames (seconds) */

# define FIRST_FRAME 0
# define MID_FRAME   1
# define LAST_FRAME  2

double x[MAX_POINTS], v[MAX_POINTS], a[MAX_POINTS], j[MAX_POINTS];

int16
   frame_index = LAST_FRAME,
   map[] = {0, 1},
   axes = (sizeof(map) / sizeof(int16));


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;
   }
}

int16 end_of_motion(int16 axes, int16 * map)
{
    int16 i;

    for(i = 0; i < axes; i++)
    {
       if(!motion_done(map[i]))
            return 0;
    }
    return 1;
}

void display(int16 axes, int16 * map)
{
    int16 i;
    double cmd;

    for(i = 0; i < axes; i++)
    {
       get_command(map[i], &cmd);
         printf("%d:%8.0lf ", map[i], cmd);
    }
    printf("\r");
}
```

```
int16 load_frames(double * cmd, double * vel, double * acc, double * jerk,
   double time)
{
   int16 i;
   FRAME frame;

   if (fifo_space() < BUFFER_SIZE)          /* Is there enough space? */
       return 1;

   if (frame_index == FIRST_FRAME)
       frame_index = MID_FRAME;
   if (frame_index == LAST_FRAME)
       frame_index = FIRST_FRAME;

   for (i = 0; i < axes; i++)
   {   if ((vel[i] == 0.0) && (acc[i] == 0.0) && (jerk[i] == 0.0))
           frame_index = LAST_FRAME;
   }

   switch (frame_index)
   {
       case FIRST_FRAME:
       {
           for (i = 0; i < axes; i++)
           {   set_gate(map[i]);
               frame_m(&frame, "0 l xvajt cund", map[i], cmd[i], vel[i],
                   acc[i], jerk[i], time, FCTL_DEFAULT | FCTL_HOLD | 0x16,
                   FUPD_POSITION | FUPD_VELOCITY | FUPD_ACCEL | FUPD_JERK |
                   FTRG_TIME, CHECK_FRAMES);
           }
           reset_gates(axes, map);   /* ready, set, go! */
           break;
       }

       case MID_FRAME:
       {
           for (i = 0; i < axes; i++)
               frame_m(&frame, "0 l xvajt cund", map[i], cmd[i], vel[i],
                   acc[i], jerk[i], time, FCTL_DEFAULT | 0x16, FUPD_POSITION |
                   FUPD_VELOCITY | FUPD_ACCEL | FUPD_JERK | FTRG_TIME,
                   CHECK_FRAMES);
           break;
       }

       case LAST_FRAME:
       {
           for (i = 0; i < axes; i++)
           {   frame_m(&frame, "0 l xvajt cund", map[i], cmd[i], vel[i],
                   acc[i], jerk[i], (1.0 / dsp_sample_rate()), FCTL_DEFAULT |
                   0x16, FUPD_POSITION | FUPD_VELOCITY | FUPD_ACCEL |
                   FUPD_JERK | FTRG_TIME, 0);
               /* set the last command position variable */
               dsp_set_last_command(dspPtr, map[i], cmd[i]);
           }
           break;
       }
   }

   return 0;
}
```

pvt4.c

Generate multi-axis coord motion profile using frames

```
void load_points(int16 n_points, double * x, double * v, double * a, double * j,
   double time)
{
   int16 i, p = 0, axis;
   double
        cmd_[PCDSP_MAX_AXES],
        vel_[PCDSP_MAX_AXES],
        acc_[PCDSP_MAX_AXES],
        jerk_[PCDSP_MAX_AXES];

   while (p < n_points)
   {
       for (i = 0; i < axes; i++)
       {   cmd_[i] = x[p];
           vel_[i] = v[p];
           acc_[i] = a[p];
           jerk_[i] = j[p];
       }

       if (load_frames(cmd_, vel_, acc_, jerk_, time))
           display(axes, map);
       else
           p++;
   }
}

void calc_points(double * x, double * v, double * a, double * j, double time)
{
   int16 i;

   /* Calculate points to generate a simple trapezoidal profile motion. */
   j[0] = 0.0;
   a[0] = ACCEL;
   v[0] = 0.0;
   x[0] = 0.0;

   j[1] = 0.0;
   a[1] = 0.0;
   v[1] = a[0] * time;
   x[1] = .5 * a[0] * time * time;

   j[2] = 0.0;
   a[2] = (-ACCEL);
   v[2] = v[1];
   x[2] = x[1] + (v[1] * time);

   j[3] = 0.0;
   a[3] = 0.0;
   v[3] = 0.0;
   x[3] = x[2] - (.5 * a[2] * time * time);

   for (i = 0; i < MAX_POINTS; i++)
       printf("\n%d J:%8.0lf A:%8.0lf V:%8.0lf X:%8.0lf", i, j[i], a[i], v[i], x[i]);
}
```

```
int16 main()
{
   int16 error_code;

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

   printf("\nCalculating points...");
   calc_points(x, v, a, j, TIME);
   printf("\ndone.\n");

   printf("\nHit any key to move...\n");
   getch();
   load_points(4, x, v, a, j, TIME);

   while(!end_of_motion(axes, map))
       display(axes, map);

    return 0;
}
```

pvu5.c

Looping frame sequence to update positions & velocities

Host Motion Profile Trajectory Generation

```
/* PVU5.C

:Looping frame sequence to update positions and velocities every 'n' samples.

This sample demonstrates how to update an axis' command velocity and command position very
quickly.  The motion profile is generated one point ahead of the current motion.  The points are
equally spaced in time.

First, a frame is downloaded to each axis in the DSP's frame buffer.  Then each frame's next
pointer is adjusted so that it points to itself.  Each frame is then executed by the DSP every 'n'
sample(s) based on the trigger/update register.  The last axis' frame is configured to generate an
interrupt to the host.

The command velocity (16 bit whole portion) and command position (32 bit whole portion) is updated
by writing directly to the frame with dsp_write_dm(...).

The command velocity is based on 3 16-bit registers (whole, upper fractional, and lower frac-
tional).  The units of the whole part are in terms of encoder counts per sample.  The default sam-
ple rate is 1250 samples per second.

The command position is based on 4 16-bit registers (upper whole, lower whole, upper fractional,
and lower fractional).  The units of the whole part are in terms of encoder counts.

At the end of the point updates, E-Stop's are generated and all of the axes decelerate to a stop.
Then the looping frames are cleared from the DSP's axis lists.

When programming with interrupt routines, make sure to define CRITICAL and ENDCRITICAL.  These are
located in idsp.h.  CRITICAL is used to disable interrupts and ENDCRITICAL re-enables interrupts.
This is very important! Interrupts during communication can cause strange problems.

If you are using a Borland C compiler (3.1 or greater), then CRITICAL and ENDCRITICAL are defined
for you.

When compiling code with an interrupt routine, to turn stack checking off.

If you modify dsp_interrupt(...), watch out for floating point operations, many compilers do not
support them.  Also, MEI's standard function library is not supported inside interrupt routines.

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 Borland and Microsoft compilers.

Written for Version 2.5
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <conio.h>
# include <dos.h>
```

```
# include <stdlib.h>
# include <math.h>
# include "pcdsp.h"
# include "idsp.h"


# define POINTS  1000

# define SAMPLE_RATE        1250          /* samples per second */
# define NEXT_UPDATE        10            /* number of samples for next update */

/* Frame register offsets - registers are 16 bits */
# define CONTROL  0x1
# define TIME_0      0x2
# define TIME_1      0x3
# define JERK_0      0x4
# define JERK_1      0x5
# define JERK_2      0x6
# define ACCEL_0  0x7
# define ACCEL_1  0x8
# define ACCEL_2  0x9
# define VEL_0    0xA
# define VEL_1    0xB
# define VEL_2    0xC
# define POSITION_0  0xD
# define POSITION_1  0xE
# define POSITION_2  0xF
# define POSITION_3  0x10
# define TRIG_UPDATE 0x11
# define ACTION      0x12
# define OUTPUT      0x13

# define R_8259_EOI  0x20
# define R_8259_IMR  0x21
# define EOI         0x20
# define EF_NONE     (-1)

# define IRQ_LINE    5

long whole_pos[PCDSP_MAX_AXES] = {0, 0, 0, 0, 0, 0, 0, 0};
int16  whole_vel[PCDSP_MAX_AXES] = {1, -1, 1, -1, 1, 1, 1, 1};      /* cts per sample */

int16 frame_addr[PCDSP_MAX_AXES];
int16 whole_vel_addr[PCDSP_MAX_AXES];
int16 whole_pos_addr[PCDSP_MAX_AXES];

typedef void (interrupt * INTERRUPT_HANDLER)();

void install_dsp_handler(int16 intno);
typedef void (interrupt * INTERRUPT_HANDLER)();
INTERRUPT_HANDLER old_handler = NULL;
static int16 imr, interrupt_number;

int16 axis_interrupts[PCDSP_MAX_AXES];
int16 update_flag = FALSE;


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;
      }
}

static int16 Read_DSP(unsigned addr)
{  outpw(dspPtr->address,addr | 0x8000);
   return(inpw(dspPtr->data));
}

static void Write_DSP(unsigned addr, int16 dm)
{
   outpw(dspPtr->address,addr | 0x8000);
   outpw(dspPtr->data,dm);
}

static int16 inc_acks(int16 axis)
{
    int16 addr1,addr2;
    int16 v1, v2 ;

    addr1 = dspPtr->global_data + axis + GD_SIZE;          /* irq_count */
    addr2 = addr1 + dspPtr->axes;                          /* ack_count */
    v1 = Read_DSP(addr1);
    v2 = Read_DSP(addr2);

    if (v1 != v2)           /* Is the DSP generating interrupts? */
      Write_DSP(addr2,v1);/* set ack_count = irq_count */
    return (v1 - v2);
}

void _far interrupt dsp_interrupt()
{
   int16 i;

   _disable();

   /* Did an axis generate the interrupt? */
   for (i = 0; i < PCDSP_MAX_AXES; i++)
   {
       if (inc_acks(i))
       {
           axis_interrupts[i]++;
           update_flag = TRUE;
       }
   }

    outp(R_8259_EOI, EOI);
}

void remove_handler(void)
{
   _disable() ;
   _dos_setvect(interrupt_number + 8, old_handler) ;
   outp(R_8259_IMR, imr) ;
   outp(R_8259_EOI, EOI) ;
   _enable() ;

}
```

```
void install_dsp_handler(int16 intno)
{
   int16 new_imr;

   interrupt_number = intno ;
   old_handler = _dos_getvect(interrupt_number + 8) ;
   imr = inp(R_8259_IMR) ;

   atexit(remove_handler) ;

   new_imr = imr & ~(1 << interrupt_number) ;
   _disable() ;
   _dos_setvect(interrupt_number + 8, dsp_interrupt) ;
   outp(R_8259_IMR, new_imr) ;
   outp(R_8259_EOI, EOI) ;
   _enable() ;
}

int16 end_of_motion(int16 n_axes, int16 * axes)
{
    int16 i;

    for(i = 0; i < n_axes; i++)
    {  if(in_motion(axes[i]))
            return 0;
    }
    return 1;
}

void display(int16 n_axes, int16 * axes)
{
    int16 i;
    double cmd;

    for(i = 0; i < n_axes; i++)
    {
      get_command(axes[i], &cmd);
        printf("%d:%8.0lf ", axes[i], cmd);
    }
    printf("\r");
}

void init_loop_frames(int16 n_axes, int16 * axes, DSP_DM update)
{
   int16 i, axis, addr, value;

   for (i = 0; i < n_axes; i++)
   {
       axis = axes[i];
       dsp_control(axis, FCTL_RELEASE, FALSE);
       dsp_control(axis, FCTL_HOLD, TRUE);
       set_gate(axis);

       dsp_marker(axis, &addr);             /* download a frame to the DSP */
       dsp_write_dm(addr, addr);            /* have the frame point to itself */

       dsp_write_dm(addr + TIME_0, NEXT_UPDATE);    /* time trigger */
       dsp_write_dm(addr + TRIG_UPDATE, update);

       frame_addr[i] = addr;
       whole_vel_addr[i] = addr + VEL_2;
       whole_pos_addr[i] = addr + POSITION_2;

       dsp_control(axis, FCTL_RELEASE, TRUE);
       dsp_control(axis, FCTL_HOLD, FALSE);
   }

   /* Configure the last axis to generate interrupts to the host. */
   addr = frame_addr[n_axes - 1] + CONTROL;
   value = dsp_read_dm(addr);
```

```
   dsp_write_dm(addr, value | FCTL_INTERRUPT);
}

void update_frames(int16 n_axes, int16 * axes, int16 * vel, long * pos)
{
   int16 i;
   int16 upper_pos, lower_pos;

   for (i = 0; i < n_axes; i++)
   {
       lower_pos = 0xFFFF & pos[i];
       upper_pos = pos[i] >> 16;
       dsp_write_dm(whole_vel_addr[i], vel[i]);
       dsp_write_dm(whole_pos_addr[i], lower_pos);
       dsp_write_dm(whole_pos_addr[i] + 1, upper_pos);
   }
   reset_gates(n_axes, axes);/* ready, set, go! */
}

void download_point(int16 n_axes, int16 * axes)
{
    int16 i;

   /* Wait for the previous point to execute before calculating the next
       point.  Since the last frame generates an interrupt, we only need to
       check the interrupt flag.
   */

   while (!update_flag)
       ;
   update_flag = FALSE;
   set_gates(n_axes, axes);/* ready, set, wait! */

   for(i = 0; i < n_axes; i++)
   {
       /* Calculate the next point.  In this sample we simply calculate the
           next point based on constant velocity.
       */
       whole_pos[i] += (whole_vel[i] * NEXT_UPDATE);
   }
   update_frames(n_axes, axes, whole_vel, whole_pos);
}

void initialize(int16 n_axes, int16 * axes)
{
   int16 error_code, axis;
   DSP_DM update;

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

    set_sample_rate(SAMPLE_RATE);

    /* Be sure to connect bit #23 to ground (enables host interrupts) */
    init_io(2, IO_INPUT);

    init_loop_frames(n_axes, axes, FUPD_POSITION | FUPD_VELOCITY | FTRG_TIME);
    install_dsp_handler(IRQ_LINE);
}
```

```
void release_frame(int16 n_axes, int16 * axes)
{
   int16 i, axis;

   for (i = 0; i < n_axes; i++)
   {
       axis = axes[i];
       set_gate(axis);
       dsp_write_dm(frame_addr[i], 0);
       dsp_write_dm(frame_addr[i] + 1, FCTL_RELEASE | FCTL_HOLD);
       dsp_write_dm(frame_addr[i] + ACTION, 0);
       dsp_write_dm(frame_addr[i] + TRIG_UPDATE, 0);
       reset_gate(axis);
   }

   for (i = 0; i < n_axes; i++)
   {
       axis = axes[i];
       while(axis_status(axis) & FRAMES_LEFT)
           ;
       while(clear_status(axis))
           ;
   }
}

int16 main()
{
   int16 i, axis, n_axes, axes[] = {0, 1, 2, 3, 4, 5};
   long point;
   DSP_DM update;

   n_axes = (sizeof(axes)/sizeof(int16));
   initialize(n_axes, axes);
   reset_gates(n_axes, axes);              /* ready, set, go! */

   for (point = 1; point < POINTS; point++)
   {   download_point(n_axes, axes);
       display(n_axes, axes);
   }

   for (i = 0; i < n_axes; i++)
       set_e_stop(axes[i]);

   while (!end_of_motion(n_axes, axes))
   {   printf("\n");
       display(n_axes, axes);
   }
   release_frame(n_axes, axes);

    return 0;
}
```

qmvs4.c

Download multiple frame sequences & execute them

Frame Sequences

```
/* QMVS4.C

:Download multiple frame sequences and selectively execute them.

This sample shows how to download a fixed set of trapezoidal profile motions and switch between
them by adjusting the next pointer in the dsp_goto(...) frames.

Here are the steps to execute several frame sequences:

 1) Download several frame sequences.  Each motion frame sequence starts with a
   "dsp_marker(...)" frame and ends with a "dsp_goto(...)" frame.  Be sure to
   turn the hold bit on in the first frame of each sequence.
 2) Set the "next" pointer in the "dsp_goto(...)" frame to select the next
   sequence.
 3) Toggle the "gate" to execute the motion.

To destroy the frame sequences, first call connect_sequence(...).  This will connect all of the
frame sequences together into one long sequence.  Then call stop_frame_loop(...).  This function
stops the motion and prevents the frames from executing.  Motion can be stopped immediately or
with the last frame.

Then call disable_frame_loop(...).  This function disables the frames, breaks the frame loop, and
releases the frames to the free list:

 1) Re-write the control, trigger/update, and action fields.  This is to
    guarantee that the DSP can step through the frames without executing them.
 2) Turn on the Hold bit in the second frame.  Set the gate.  This will force
    the DSP to halt at the first frame.
 3) Clear the status.
 4) Wait for the DSP to reach the first frame.
 5) Turn on the release bit in each frame.
 6) Break the loop by setting the last frame's 'next' pointer to zero.
 7) Reset the gate flag.  The DSP will step through the frames one by one and
    release each one to the free list.
 8) Wait for all of the frames to be released to the free list.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include <dos.h>
# include "pcdsp.h"
# include "idsp.h"
```

```
# define SAMPLE_RATE 1250/* samples per second */

# define AXIS    0
# define MOVES   3          /* number of moves */
# define MAX_FRAMES  10

double distance[MOVES] = {4096.0, 8192.0, -32768.0};
double velocity[MOVES] = {10000.0, 10000.0, 10000.0};
double accel[MOVES] = {500000.0, 500000.0, 500000.0};

int16 start[MOVES];
int16 end[MOVES];

/* Frame register offsets - registers are 16 bits */
# define CONTROL  0x1
# define TRIG_UPDATE    0x11
# define ACTION         0x12


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;
   }
}

void display(int16 axis)
{
   double cmd;

   get_command(axis, &cmd);
   printf("\rCmd:%8.0lf", cmd);
}

void go(int16 axis)
{
   int16 sample_time;

   reset_gate(axis);  /* ready, set, go! */

   dsp_write_dm(0x100, 0x0);        /* write to DSP's signature word */
   while (!dsp_read_dm(0x100))      /* wait for DSP to write signature */
       ;
   dsp_write_dm(0x100, 0x0);
   while (!dsp_read_dm(0x100))
       ;

   set_gate(axis);           /* prevent sequence from executing twice */
}
```

```
int16 set_first_move(int16 axis, int16 next, int16 * start)
{
   int16 addr;

   /* Set the 'next' pointer in the place holder frame. */
   addr = dsp_read_dm(dspPtr->outptr + axis);
   dsp_write_dm(addr, start[next]);

   return next;
}

int16 set_next_move(int16 current, int16 next, int16 * start, int16 * end)
{
   /* Set the 'next' pointer in the dsp_goto(...) frame. */
   dsp_write_dm(end[current], start[next]);

   return next;
}

void connect_sequence(int16 * start, int16 * end)
{
   int16 i, next;

   /* Connect all of the frame sequences together. */
   for (i = 0; i < MOVES; i++)
   {
       next = (i + 1) % MOVES;
       dsp_write_dm(end[i], start[next]);
   }
}

int16 stop_frame_loop(int16 axis, int16 last_frame, int16 immediate)
{
   if (immediate)
       set_e_stop(axis);
   else
       dsp_write_dm(last_frame + ACTION, E_STOP_EVENT);

   /* Wait for the E-Stop Event to complete */
   while((axis_state(axis) != E_STOP_EVENT) || in_motion(axis))
       ;

   return dsp_error;
}

int16 disable_frame_loop(int16 axis, int16 last_frame)
{
   int16 i, control, first_frame, fp, fp2;

   fp = first_frame = dsp_read_dm(last_frame);
   fp2 = dsp_read_dm(fp);           /* address of second frame */

   /* Disable the looping frames */
   do
   {   dsp_write_dm(fp + CONTROL, (FCTL_DEFAULT & ~FCTL_RELEASE));
       dsp_write_dm(fp + TRIG_UPDATE, 0);
       dsp_write_dm(fp + ACTION, 0);
       fp = dsp_read_dm(fp); /* address of next frame */
   }
   while(fp != first_frame);

   /* Turn on the hold bit in the second frame */
   dsp_write_dm(fp2 + CONTROL, (FCTL_DEFAULT & ~FCTL_RELEASE | FCTL_HOLD));
   set_gate(axis);
   while(clear_status(axis))
       ;

   /* wait for the DSP to reach the first frame */
   while (first_frame != (dsp_read_dm(dspPtr->outptr + axis)))
       ;
```

```
   fp = first_frame;

   /* Turn on the release bit in the looping frames */
   do
   {   control = dsp_read_dm(fp + CONTROL);
       dsp_write_dm(fp + CONTROL, (control | FCTL_RELEASE));
       fp = dsp_read_dm(fp);        /* address of next frame */
   }
   while(fp != first_frame);

   dsp_write_dm(last_frame, 0x0);/* break the loop */
   reset_gate(axis);               /* release the frames */

   while(axis_status(axis) & FRAMES_LEFT)
       ;

   return dsp_error;
}

int16 initialize_frames(int16 axis, int16 * start, int16 * end)
{
   int16 i, j, control, last_frame, frame_addr[MAX_FRAMES];

   set_gate(axis);/* prevent frames from executing */

   for (i = 0; i < MOVES; i++)
   {
       dsp_control(axis, FCTL_RELEASE, FALSE);              /* turn off the release bit */
       dsp_control(axis, FCTL_HOLD, TRUE);                  /* turn on the hold bit */
       frame_m(NULL, "0l * n d", axis, start + i, 0);       /* place holder */
       dsp_control(axis, FCTL_HOLD, FALSE);                 /* turn off the hold bit */

       start_r_move(axis, distance[i], velocity[i], accel[i]);

       dsp_goto(axis, 0x0);  /* We don't know where to go yet! */

       /* Turn off the hold bit in the frames, find the last frame. */
       j = 1;
       frame_addr[j] = dsp_read_dm(start[i]);
       while ((dsp_read_dm(frame_addr[j])) && (j < MAX_FRAMES))
       {
           frame_addr[j+1] = dsp_read_dm(frame_addr[j]);
           control = dsp_read_dm(frame_addr[j] + CONTROL);
           dsp_write_dm(frame_addr[j] + CONTROL, control & ~FCTL_HOLD);
           j++;
       }
       last_frame = end[i] = frame_addr[j];
   }
   dsp_control(axis, FCTL_RELEASE, TRUE);   /* turn the release bit back on */

   return last_frame;
}

void initialize(void)
{
   int16 error_code;

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

   set_sample_rate(SAMPLE_RATE);
}
```

qmvs4.c

```
int16 main()
{
   int16 done, key, current, next, last_frame, first_move = 1;

   initialize();
   last_frame = initialize_frames(AXIS, start, end);
   current = 0;   /* first move type */

   printf("\n\n(0-%d)=move type, g=go, d=disable frames, esc=quit\n", MOVES-1);

   for (done = 0; !done; )
   {
       display(AXIS);

       if (kbhit())     /* key pressed? */
       {   key = getch() ;

           switch (key)
           {
              case 'g':
                  go(AXIS);  /* Be sure the next move is defined! */
                  first_move = FALSE;
                  break;

              case 'd':
                  connect_sequence(start, end);
                  stop_frame_loop(AXIS, last_frame, TRUE);
                  disable_frame_loop(AXIS, last_frame);
                  printf("\nFrame loop disabled.\n");
                  break;

              case 0x1B:      /* <ESC> */
                  done = TRUE;
                  break;

              default:
                  next = (key - 48);
                  if ((next >= 0) && (next < MOVES))
                  {
                      printf("\nCurrent: %d Next: %d Distance: %8.0lf \n",
                             current, next, distance[key-48]);

                      if (first_move)
                             current = set_first_move(AXIS, next, start);
                      else
                             current = set_next_move(current, next, start, end);
                  }
                  break;
           }
       }
   }
    return 0;
}
```

Download multiple frame sequences & execute them

redfed.c

Redundant feedback checking

Redundant Feedback

```
/* REDFED.C

:Redundant feedback checking.

This code demonstrates how to configure the DSP to check for invalid feedback based on two feed-
back devices.  The first feedback device is connected to the "Real" axis.  The second feedback
device is connected to the "Redundant" axis.  Motion is commanded on the "Real" axis.  Both feed-
back devices are connected to the same mechanical system.

Here are the steps to configure feedback checking:

 1) Set the "Real" and "Redundant" actual positions equal
 2) Positionally link the "Real" and "Redundant" axes
 3) Configure the error limits and event responses for the axes
 4) Configure the "Redundant" axis to send an exception event to the "Real" axis
   when an exception event occurs

If the difference between "Real" axis' command position and the actual position exceed the error
limit, an exception event will be generated on the "Real" axis.  If the difference between the
actual positions of the axes exceed the error limit an exception event will be generated on the
"Redundant" axis. If an exception event occurs on the "Redundant" axis, an event will be sent to
the "Real" axis.  The DSP handles the error limit checking and exception event generation.

The Stop, E-Stop, and Abort Events are executed by the DSP based on an axis' limit switches, home
switch, software limits, error limits, etc.  Internally, the DSP executes some special frames to
complete the exception event.

When a Stop Event or E-Stop Event is generated, the DSP executes a Stop or E-Stop frame to decel-
erate the axis.  Then the DSP executes the Stopped frame to set the command velocity to zero.  An
Abort Event puts the axis in idle mode (no PID update) and executes the Stopped frame.  See the "C
Programming" manual for more information about Exception Events.

By modifying an axis' Stopped exception frame, an exception event can be sent to another axis by
the DSP after a Stop, E-Stop, or Abort Event executes.

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

*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"
# include "idsp.h"

# define REAL_AXIS   0
```

```
# define REDUNDANT_AXIS     1

# define RATIO              (-1.0)
# define ERROR_LIMIT        20.0
# define ACT_DIFF_LIMIT     30.0


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;
   }
}

void display(int16 axis1, int16 axis2)
{
   int16 state1, state2;
   double act1, act2;

   while (!kbhit())
   {
       get_position(axis1, &act1);
       get_position(axis2, &act2);
       state1 = axis_state(axis1);
       state2 = axis_state(axis2);
       printf("Pos: %8.0lf State: %d Pos: %8.0lf State: %d\r", act1, state1,
           act2, state2);
   }
   getch();
}

int16 config_stopped_frame(PFRAME f, int16 address, int16 ormask, int16 andmask)
{
   int16 update = f->f.trig_update & 0xFF;/* get trigger/update field */
   int16 * i = (int16*) &(f->f.position);/* pointer to position field */

   if (!(update & FUPD_POSITION))
   {
       f->f.output = OUTPUT_POSITION ;      /* set output to use position field */
       i[0] = address;
       i[1] = ormask;
       i[2] = andmask;
       f->f.trig_update |= FUPD_OUTPUT;     /* update register from position field */
       pcdsp_write_ef(dspPtr, f);           /* download the exception frame */
       return DSP_OK ;
   }
   return DSP_RESOURCE_IN_USE;
}
```

```
int16 set_stopped_frame(int16 axis, int16 destaxis, int16 event)
{
   FRAME f;

   event |= (ID_AXIS_COMMAND << 4);

   if (pcdsp_sick(dspPtr, axis) || pcdsp_sick(dspPtr, destaxis))
       return dsp_error;

   if (pcdsp_read_ef(dspPtr, &f, axis, EF_STOPPED))
       return dsp_error;

   if (config_stopped_frame(&f, dspPtr->pc_event + destaxis, event, event))
       return dsp_error;
   else
       return pcdsp_write_ef(dspPtr, &f);
}

int16 main()
{
   int16 error_code;

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

   set_position(REAL_AXIS, 0.0);    /* synchronize position registers */
   set_position(REDUNDANT_AXIS, 0.0);

   set_positive_sw_limit(REDUNDANT_AXIS, 100000.0, NO_EVENT);
   set_negative_sw_limit(REDUNDANT_AXIS, 100000.0, NO_EVENT);
   set_error_limit(REAL_AXIS, ERROR_LIMIT, ABORT_EVENT);
   set_error_limit(REDUNDANT_AXIS, ACT_DIFF_LIMIT, ABORT_EVENT);

   set_stopped_frame(REDUNDANT_AXIS, REAL_AXIS, ABORT_EVENT);

   mei_link(REAL_AXIS, REDUNDANT_AXIS, RATIO, LINK_ACTUAL);
   display(REAL_AXIS, REDUNDANT_AXIS);

   endlink(REAL_AXIS);

    return 0;
}
```

rtraj.c

Read real-time trajectory info from DSP

Data Acquisition

```
/* RTRAJ.C

:Read real-time trajectory information from the DSP.

This code shows how to use get_tuner_data(...) to sample the command positions, actual positions,
error, time, state, and dac output.  Any motion can be commanded while the real-time response is
sampled.

This feature is very useful in examining the system response to determine optimum tuning parame-
ters.

Note: get_tuner_data(...) is an "extra" function and is not documented in the "C Programming" man-
ual.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <dos.h>
# include <conio.h>
# include "pcdsp.h"
# include "extras.h"


# define AXIS               0
# define BUFFER_SECONDS     5


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;
   }
}
```

rtraj.c

Read real-time trajectory info from DSP

```
int16 main()
{
   int16 i, error_code;
   size_t buffer_length;

   long
       * apos,
       * cpos;
   int16
       * time,
       * state,
       * voltage;

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

   buffer_length = BUFFER_SECONDS * dsp_sample_rate();

   apos = calloc(buffer_length, sizeof(apos[0]));
   cpos = calloc(buffer_length, sizeof(cpos[0]));
   time = calloc(buffer_length, sizeof(time[0]));
   state = calloc(buffer_length, sizeof(state[0]));
   voltage = calloc(buffer_length, sizeof(voltage[0]));

   if (!(apos && cpos && time && state && voltage))
   {   fprintf(stderr, "Not enough memory.\n");
       return 1;
   }

   fprintf(stderr, "Press any key to begin.\n");
   getch();

   fprintf(stderr, "Sampling...");

   /*  ************************************************  **
       put whatever type of motion you want to sample here.
   **  ************************************************  */

   get_tuner_data(AXIS, buffer_length, apos, cpos, time, state, voltage);

   fprintf(stderr, "done.\n");

   printf("apos\tcpos\ttime\tstate\tvoltage\n");
   for (i = 0; i < buffer_length; i++)
   {   printf("%ld\t%ld\t%u\t%d\t%d\n",
           apos[i], cpos[i], time[i], state[i], voltage[i]);
   }

   return 0;
}
```

sattr.c

SERCOS: Read IDN attributes

SERCOS

```
/* SATTR.C

:SERCOS IDN attributes.

This sample program demonstrates how to read an IDN's attributes from a SERCOS node.  The
attributes consist of 7 elements defined in the SERCOS specification:

Element 1:
 "ID Number"
 size is 16 bits, binary
 mandatory

Element 2:
 "Name of operation data"
 maximum size is 64 bytes
 upper 4 bytes specify the string size
 lower 60 bytes are the string
 optional

Element 3:
 "Attribute of operation data"
 size is 4 bytes, binary
 bits 0-15 represent the conversion factor
 bits 16-18 represent the data length
 bit 19 represents the function of operation data
 bits 20-22 represent the data type and display format
 bits 24-27 represent places after the decimal point
 bits 28-31 reserved
 mandatory

Element 4:
 "Operation data unit"
 maximum size is 16 bytes, binary/string
 bytes 1-2 specify length of programmed text in drive
 bytes 3-4 indicate maximum text length (in bytes) in the drive
 bytes 4-12 variable length character text string
 optional

Element 5:
 "Minimum input value of operation data"
 size is the same as Element 7 (Operation Data), binary
 optional

Element 6:
 "Maximum input value of operation data"
 size is the same as Element 7 (Operation Data), binary
 optional

Element 7:
 "Operation data"
 maximum size is 65532 bytes, binary and/or string
 Fixed length:
   bytes 1-2, operation data, 2 bytes
   bytes 1-4, operation data, 4 bytes
 Variable length:
   bytes 1-2, specify length of programmed data (bytes) in drive
   bytes 3-4, indicate maximum data length available (bytes) in drive
   bytes 5-length, operation data
```

```
Be sure to initialize the SERCOS communication ring with serc_reset(...)
 before reading an IDN's attributes.

Written for Motion Library Version 2.5
*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "pcdsp.h"
#include "sercos.h"

#define AXIS                0       /* Controller Axis */
#define DRIVE_ADDRESS       1       /* SERCOS Node Address */
#define IDN                 79      /* Rotational Position Resolution */


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;
   }
}
```

sattr.c

```
int main(void)
{
   int16 error_code, size = 0;
   char * endp;
   long *buff, exp, proc, type;
   IDN_ATTRIBUTES attr;

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

   error(get_idn_attributes(AXIS, IDN, &attr));

   printf("\nidn: %u (0x%X)\n", IDN, IDN);
   printf("%s-%d-%d\n", attr.elem_1 & 0x8000 ? "P":"S",
       (attr.elem_1 & 0x7000)>>12, attr.elem_1 & 0xFFF);

   printf("%s\n", attr.elem_2);

   if(((attr.elem_3>>16) & 7) == 1)
       size = 2;
   if(((attr.elem_3>>16) & 7) == 2)
       size = 4;
   if(size)
       printf("Size: %d bytes\n", size);
   else
       printf("Size: Variable length string\n");

   proc = attr.elem_3 & 0x80000;
   printf("Data Type: %s\n", proc?"Procedure Command":"Operation Data or Parameter");
   if(!proc)
   {   exp = (attr.elem_3>>24) & 0xF;
       printf("Conversion Factor: %ldx10e%ld\n", (attr.elem_3&0xFFFF), exp);
       printf("Units: %s\n", attr.elem_4);
       printf("Minimum Value: %ld\n", attr.elem_5);
       printf("Maximum Value: %ld\n", attr.elem_6);
   }

   return 0;
}
```

scanadp.c

Trapezoidal profile motion: Capture values at positions

Data Acquisition

```
/* SCANADP.C

:Trapezoidal profile motion and capture analog values at specific positions.

This code configures the DSP to generate a trapezoidal profile motion and capture analog values at
specific positions.

The actual position (analog value) capturing is handled automatically by the DSP.  After each
frame is executed, the DSP copies the actual position into the frame and releases it to the free
list.  Later, the captured positions are retrieved from the frames.  There are 600 frames in the
DSP's external memory.

The steps are:
 1) Configure an axis for analog feedback
 2) Download a trapezoidal profile motion
 3) Download a sequence of position triggered frames and read their addresses.
    The addresses will be needed later to read the captured positions
 4) Read the captured positions (analog values) from the frames

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"
# include "idsp.h"

# define MOVE_AXIS          0
# define ANALOG_AXIS        2
# define ANALOG_CHAN        0

# define DISTANCE           30000.0
# define VELOCITY           10000.0
# define ACCEL              100000.0

# define CAPTURES           30
# define DIST_INTERVAL      1000.0         /* units = encoder counts */

# define RELEASE_POS_2      0x2
# define RELEASE_POS_3      0x3

int16 scan_addr[CAPTURES];
long scan_pos[CAPTURES];
```

```
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;
   }
}

void load_scan_frames(int16 axis, int16 taxis, int16 sense, int16 captures, double interval)
{
   int16 i;
   double tpos;

   tpos = interval;

   dsp_control(axis, FCTL_HOLD, TRUE);
   dsp_position_trigger(axis, taxis, tpos, sense, TRUE, NEW_FRAME);
   scan_addr[0] = dsp_read_dm(dspPtr->inptr + axis);               /* frame address */
   dsp_control(axis, FCTL_HOLD, FALSE);

   for (i = 1; i < captures; i++)
   {   tpos += interval;
       dsp_position_trigger(axis, taxis, tpos, sense, TRUE, NEW_FRAME);
       scan_addr[i] = dsp_read_dm(dspPtr->inptr + axis);           /* frame address */
   }
   dsp_end_sequence(axis);
}

void read_positions(int16 captures)
{
   int16 i;
   unsigned16 lower;
   long upper;

   for (i = 0; i < captures; i++)
   {
       lower = dsp_read_dm(scan_addr[i] + RELEASE_POS_2);
       upper = dsp_read_dm(scan_addr[i] + RELEASE_POS_3);
       scan_pos[i] = (lower | (upper << 16));
       printf("\nAddr: 0x%x Scan: %ld", scan_addr[i], scan_pos[i]);
   }
}
```

```
int16 main()
{
   int16 error_code, axes[] = {MOVE_AXIS, ANALOG_AXIS};
   double position;

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

   /* Configure an axis for analog feedback. */
   set_feedback(ANALOG_AXIS, FB_ANALOG);
   set_analog_channel(ANALOG_AXIS, ANALOG_CHAN, FALSE, FALSE);

   set_gates(2, axes);              /* Prevent frame execution */
   load_scan_frames(ANALOG_AXIS, MOVE_AXIS, POSITIVE_SENSE, CAPTURES, DIST_INTERVAL);
   start_r_move(MOVE_AXIS, DISTANCE, VELOCITY, ACCEL);
   reset_gates(2, axes);            /* ready, set, go! */
   printf("\nScanning positions\n");

   while (!motion_done(MOVE_AXIS) || !motion_done(ANALOG_AXIS))
   {   get_command(MOVE_AXIS, &position);
       printf("\rPosition: %8.0lf ", position);
   }
   read_positions(CAPTURES);

    return 0;
}
```

scanadt.c

Trapezoidal profile motion: Capture values at times

Data Acquisition

```
/* SCANADT.C

:Trapezoidal profile motion and capture analog values at specific times.

This code configures the DSP to generate a trapezoidal profile motion and capture analog values at
specific time intervals.  The minimum time interval is one sample (800 microseconds for the
default sample rate).

The actual position (analog value) capturing is handled automatically by the DSP.  After each
frame is executed, the DSP copies the actual position into the frame and releases it to the free
list.  Later, the captured positions are retrieved from the frames.  There are 600 frames in the
DSP's external memory.

The steps are:
 1) Configure an axis for analog feedback
 2) Download a trapezoidal profile motion
 3) Download a sequence of dwell frames and read their addresses.
    The addresses will be needed later to read the captured positions.
 4) Read the captured positions (analog values) from the frames

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"
# include "idsp.h"

# define MOVE_AXIS          0
# define ANALOG_AXIS        2
# define ANALOG_CHAN        0

# define DISTANCE    30000.0
# define VELOCITY    10000.0
# define ACCEL       100000.0

# define CAPTURES           30
# define TIME_INTERVAL      .1      /* units = seconds, minimum is one DSP sample */

# define RELEASE_POS_2      0x2
# define RELEASE_POS_3      0x3

int16 scan_addr[CAPTURES];
long scan_pos[CAPTURES];
```

```
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;
   }
}

void load_scan_frames(int16 axis, int16 captures, double interval)
{
   int16 i;

   dsp_control(axis, FCTL_HOLD, TRUE);
   dsp_dwell(axis, interval);
   scan_addr[0] = dsp_read_dm(dspPtr->inptr + axis);        /* frame address */
   dsp_control(axis, FCTL_HOLD, FALSE);

   for (i = 1; i < captures; i++)
   {   dsp_dwell(axis, interval);
       scan_addr[i] = dsp_read_dm(dspPtr->inptr + axis);    /* frame address */
   }
   dsp_end_sequence(axis);
}

void read_positions(int16 captures)
{
   int16 i;
   unsigned16 lower;
   long upper;

   for (i = 0; i < captures; i++)
   {
       lower = dsp_read_dm(scan_addr[i] + RELEASE_POS_2);
       upper = dsp_read_dm(scan_addr[i] + RELEASE_POS_3);
       scan_pos[i] = (lower | (upper << 16));
       printf("\nAddr: 0x%x Scan: %ld", scan_addr[i], scan_pos[i]);
   }
}
```

scanadt.c

```
int16 main()
{
   int16 error_code, axes[] = {MOVE_AXIS, ANALOG_AXIS};
   double position;

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

   /* Configure an axis for analog feedback. */
   set_feedback(ANALOG_AXIS, FB_ANALOG);
   set_analog_channel(ANALOG_AXIS, ANALOG_CHAN, FALSE, FALSE);

   set_gates(2, axes);/* Prevent frame execution */
   load_scan_frames(ANALOG_AXIS, CAPTURES, TIME_INTERVAL);
   start_r_move(MOVE_AXIS, DISTANCE, VELOCITY, ACCEL);
   reset_gates(2, axes);            /* ready, set, go! */
   printf("\nScanning positions\n");

   while (!motion_done(MOVE_AXIS) || !motion_done(ANALOG_AXIS))
   {   get_command(MOVE_AXIS, &position);
       printf("\rPosition: %8.0lf ", position);
   }
   read_positions(CAPTURES);

    return 0;
}
```

Trapezoidal profile motion: Capture values at times

scancap.c

Use velocity profile scan to capture positions

Data Acquisition

```
/* SCANCAP.C

:Velocity profile scan to capture positions.

This code configures the DSP to generate a velocity profile and capture positions at specific time
intervals.  The minimum time interval is one sample (800 microseconds for the default sample
rate).  The position capturing is handled automatically by the DSP.  After each frame is executed,
the DSP copies the actual position into the frame and releases it to the free list. Later, the
captured positions are retrieved from the frames.  There are 600 frames in the DSP's external mem-
ory.

The steps are:
 1) Download a velocity profile motion
 2) Download a sequence of dwell frames and read their addresses.
    These will be needed later to read the captured positions.
 3) Read the captured positions from the frames

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"
# include "idsp.h"

# define AXIS        0
# define VELOCITY    20000.0
# define ACCEL       100000.0

# define CAPTURES           100
# define TIME_INTERVAL.     01            /* units = seconds, minimum is one DSP sample */

# define RELEASE_POS_2      0x2
# define RELEASE_POS_3      0x3

int16 scan_addr[CAPTURES];
long scan_pos[CAPTURES];
```

scancap.c

Use velocity profile scan to capture positions

```
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;
   }
}

void load_profile(int16 axis, int16 captures, double interval)
{
   int16 i;

   /* Turn on the axis gate flag to prevent frame execution */
   set_gate(axis);
   v_move(axis, VELOCITY, ACCEL);

   for (i = 0; i < captures; i++)
   {
       dsp_dwell(axis, interval);
       scan_addr[i] = dsp_read_dm(dspPtr->inptr + axis);     /* frame address */
   }
   v_move(axis, 0.0, ACCEL);

   reset_gate(axis);                                         /* ready, set, go! */
}

void read_positions(int16 captures)
{
   int16 i;
   unsigned16 lower;
   long upper;

   for (i = 0; i < captures; i++)
   {
       lower = dsp_read_dm(scan_addr[i] + RELEASE_POS_2);
       upper = dsp_read_dm(scan_addr[i] + RELEASE_POS_3);
       scan_pos[i] = (lower | (upper << 16));
       printf("\nAddr: 0x%x Scan: %ld", scan_addr[i], scan_pos[i]);
   }
}
```

```
int16 main()
{
   int16 error_code;
   double position;

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

   load_profile(AXIS, CAPTURES, TIME_INTERVAL);
   printf("\nScanning positions\n");

   while (!motion_done(AXIS))
   {
       get_position(AXIS, &position);
       printf("\rPosition: %8.0lf ", position);
   }
   read_positions(CAPTURES);

    return 0;
}
```

Use velocity profile scan to capture positions

scrstat.c

SERCOS: Read ring status

SERCOS

```
/* SCRSTAT.C

:SERCOS communication ring status.

This sample demonstrates how to determine the SERCOS communication ring status.  The SERCON 410B
has two status flags which monitor the data and the ring closure.

The RDIST bit is low when data is valid, and is high when the data is distorted.

The FIBBR bit is low when the loop is closed, and is high when the loop is open.

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 Motion Library Version 2.5
*/


#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "pcdsp.h"

#define RING_STATUS_ADDR     0xC02          /* SERCON 410B Register */

#define RDIST                0x1000         /* distorted data bit */
#define FIBBR                0x2000         /* loop open bit */


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()
{
   int16 error_code, value;

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

   while (!kbhit())
   {   value = dsp_read_pm(RING_STATUS_ADDR);
       printf("\rDistort:%d Loop Open:%d",
           (value & RDIST) ? 1:0, (value & FIBBR) ? 1:0);
   }
   getch();

    return 0;
}
```

sdiag.c

SERCOS: Read drive diagnostics & reset/enable drive

SERCOS

```
/* SDIAG.C

:SERCOS drive fault diagnostics/recovery.

This sample program demonstrates how to read a SERCOS drive's Drive Status and diagnostic mes-
sages.  If the drive is shutdown due to an error in the Class 1 Diagnostic, then a drive reset/
enable is attempted.

Be sure to initialize the SERCOS communication ring with serc_reset(...) before reading the drive
diagnostic information.

Written for Motion Library Version 2.5
*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "pcdsp.h"
#include "sercos.h"

#define AXIS                0       /* Controller Axis */
#define DRIVE_ADDRESS       1       /* SERCOS Node Address */

#define CLASS_1_ERROR       0x2000          /* bit 13 */
#define CLASS_2_ERROR       0x1000          /* bit 12 */

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(void)
{
   int16 error_code, code, status;
   char message[MAX_ERROR_LEN];

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

   error(get_drive_status(AXIS, &status));
   printf("\nDrive Status:0x%x", status);

   /* Operating Status Drive Message */
   error(get_idn_string(AXIS, DRIVE_ADDRESS, 95, message));
   printf("\nDrive Msg: %s", message);

   if (status & CLASS_1_ERROR)
   {   error(disable_amplifier(AXIS));             /* disable SERCOS drive */
       error(get_class_1_diag(AXIS, &code, message));
       printf("\nClass 1:(0x%x) %s", code, message);

       error(reset_sercos_drive(AXIS));            /* clear Class 1 Diagnostic Error */
       error(controller_run(AXIS));                /* clear controller Abort Event */
       error(enable_amplifier(AXIS));              /* enable SERCOS drive */
       printf("\nDrive Re-enabled");
   }
   if (status & CLASS_2_ERROR)
   {   error(get_class_2_diag(AXIS, &code, message));
       printf("\nClass 2:(0x%x) %s", code, message);
   }

   /* Operating Status Drive Message */
   error(get_idn_string(AXIS, DRIVE_ADDRESS, 95, message));
   printf("\nDrive Msg: %s", message);

   return 0;
}
```

sem.c

Multi-tasking & interrupts under WinNT

Windows NT

```
/* SEM.C

:Multi-tasking and interrupts under Windows NT.

This sample demonstrates how to create a separate thread for each axis. The number of axes is
specified by the define AXES.  Each thread will enter an efficient wait state until an interrupt
that is generated from an event occurs.  The thread will then clear the event by calling
controller_run(...) and then go back to sleep until the next interrupt occurs.

In order for threads to use the mei_sleep_until_interrupt(...) function without error, each thread
MUST use its own HANDLE to the DSPIO driver in the function call.  The HANDLE is created with a
call to CreateFile(...).

Like mei_sleep_until_interrupt(), all other library functions must have a HANDLE to the DSPIO
driver.  This HANDLE is created when either dsp_init(...) or do_dsp(...) are called.  This HANDLE
is held by the library and is used for all library access to the MEI card.

This code uses a global semaphore (MEI_SEMAPHORE) to gate access to the library so that only one
thread is allowed to communicate with the MEI card at a time.  The semaphore is created by a call
to create_MEI_semaphore(...) and destroyed by a call to kill_MEI_semaphore(...).

All calls to the MEI library are bracketed by calls to get_MEI_semaphore(...) and
release_MEI_semaphore(...).

When using interrupts, be sure to set the appropriate IRQ switch on the DSP-Series controller.
Also, make sure the device driver "DSPIO" is configured for the same IRQ.

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 <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "idsp.h"

#ifdef MEI_MSVC20           /* support for Microsoft Visual C/C++ ver 2.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif

#ifdef MEI_MSVC40           /* support for Microsoft Visual C/C++ ver 4.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif

#define   AXES   (int16)4          /* This also defines the number of
                                      threads to be created */
#define   SEM_MAXIMUM_USAGE               1L
#define   SEM_INITIAL_SIGNALED            1L
#define   SEM_INITIAL_NON_SIGNALED        0L
#define   SEM_INCREMENT                   1L
#define   THREAD_STACK_SIZE               0
#define   THREAD_CREATION_FLAGS           0
#define   EXIT_CODE_OK                    0
```

```
/* Globals */
HANDLE       MEI_SEMAPHORE;        /* Handle to library semaphore */
HANDLE       hThreads[AXES];       /* Handles to each created task */

typedef struct interrupt_thread_args{
   int16   axis;
   HANDLE  semaphore;
}InterruptThreadArgs;
/* create one InterruptThreadArgs structure for each axis */
InterruptThreadArgs ThreadArgs[AXES];


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;
   }
}

/* Create the semaphore object.  This semaphore will allow access to the
   MEI library for one task at time.  All other tasks will wait on the
   semaphore until it is available.  The initial state of the semaphore
   is signaled. */
HANDLE create_MEI_semaphore(void)
{  return CreateSemaphore(NULL, SEM_INITIAL_SIGNALED,
       SEM_MAXIMUM_USAGE, "MEI_semaphore");
}

/* Terminate the semaphore object */
int16 kill_MEI_semaphore(HANDLE sem)
{  return (int16)CloseHandle(sem);
}

/* Wait for the semaphore to be signaled and set the semaphore to a
   non-signaled state */
int16 get_MEI_semaphore(HANDLE sem)
{  return (int16)WaitForSingleObject(sem, INFINITE);
}

/* Release the semaphore by incrementing it to a signaled state */
int16 release_MEI_semaphore(HANDLE sem)
{  return ReleaseSemaphore(sem, SEM_INCREMENT, NULL);
}
```

```
void MEI_InterruptThread(void *args)
{  InterruptThreadArgs *pArgs;
   HANDLE            thread_semaphore;
   int16                    axis;
   HANDLE            file_ptr;

   pArgs = (InterruptThreadArgs *)args;
   thread_semaphore = pArgs->semaphore;
   axis = pArgs->axis;

   /* create a separate file pointer to the driver for each task */
   file_ptr = CreateFile("\\\\.\\dspio0", GENERIC_WRITE|GENERIC_READ,
                FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL);

   /* set thread_semaphore to a signaled state so that the rest of
       the program can resume */
   ReleaseSemaphore(thread_semaphore, SEM_INCREMENT, NULL);

   while(1)
   {   mei_sleep_until_interrupt(file_ptr, axis, MEI_EVENT);
       get_MEI_semaphore(MEI_SEMAPHORE);
           while(controller_run(axis))
               ;
       release_MEI_semaphore(MEI_SEMAPHORE);
   }
}

void setup(int16 axis)
{  unsigned long ThreadId;

   get_MEI_semaphore(MEI_SEMAPHORE);
       controller_run(axis);
       set_home_level(axis, FALSE);
       set_home(axis, ABORT_EVENT);
       clear_status(axis);
       interrupt_on_event(axis, TRUE);
   release_MEI_semaphore(MEI_SEMAPHORE);


   ThreadArgs[axis].axis = axis;
   /* create a semaphore to wait for created thread to finish initializing */
   ThreadArgs[axis].semaphore = CreateSemaphore(NULL, SEM_INITIAL_NON_SIGNALED,
       SEM_MAXIMUM_USAGE, NULL);

   /* create thread
       default security descriptor, default stack size (same as creating thread)*/
   hThreads[axis] = CreateThread(NULL, THREAD_STACK_SIZE,
       (LPTHREAD_START_ROUTINE)MEI_InterruptThread, &ThreadArgs[axis],
       THREAD_CREATION_FLAGS, &ThreadId);

   /* wait for created thread to finish initializing.  This occurs when
       MEI_InterruptThread releases the thread_semaphore. */
   WaitForSingleObject(ThreadArgs[axis].semaphore, INFINITE);

   /* semaphore to created thread is no longer needed */
   CloseHandle(ThreadArgs[axis].semaphore);
}
```

```
int main()
{  int16 error_code, axis, state[AXES], source[AXES];

   MEI_SEMAPHORE = create_MEI_semaphore();

   get_MEI_semaphore(MEI_SEMAPHORE);

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

   init_io(PORT_A, IO_INPUT);       /* initialize I/O port for inputs */
   init_io(PORT_B, IO_INPUT);
   init_io(PORT_C, IO_OUTPUT);      /* initialize I/O port for outputs */

   reset_bit(23);                   /* enable interrupt generation */
   release_MEI_semaphore(MEI_SEMAPHORE);

   for(axis = 0; axis < (int16)AXES; axis++)
       setup(axis);

   while(!kbhit())
   {   for(axis = 0; axis < AXES; axis++)
       {   get_MEI_semaphore(MEI_SEMAPHORE);
           state[axis] = axis_state(axis);
           source[axis] = axis_source(axis);
           release_MEI_semaphore(MEI_SEMAPHORE);
           printf("Axis: %d state<%d> source<%d>\n",
               axis, state[axis], source[axis]);
       }
   }
   getch();
   kill_MEI_semaphore(MEI_SEMAPHORE);
   for(axis = 0; axis < AXES; axis++)
   {   CloseHandle(hThreads[axis]);
       TerminateThread(hThreads[axis], EXIT_CODE_OK);
   }
   return 0;
}
```

sercpl2.c

SERCOS: Use position latching with a drive

SERCOS

```
/* SERCPL2.C

:Position Latching with a SERCOS drive.

This sample demonstrates how to configure a SERCOS drive to latch the position based on 2 probe
inputs.  Probe 1 and 2 are configured to trigger on the rising edge.  The probe status is mapped
into the Drive Status word.  The real time status word is also copied to external memory by the
DSP every sample.

Be sure to initialize the SERCOS communication ring with serc_reset(...) before configuring the
drive for position latching.

For more information on Probe based Position Latching, please consult the drive manufacturer's
documentation.

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 Motion Library Version 2.5
*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "pcdsp.h"
#include "sercos.h"

#define AXIS         0
#define RT_STATUS_1  0x40
#define RT_STATUS_2  0x80


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()
{
   int16 error_code, drive_status;
   long val;
   double act;

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

   error(set_idn(AXIS, 405, 1));           /* Probe 1 Enable */
   error(set_idn(AXIS, 406, 1));           /* Probe 2 Enable */
   set_idn(AXIS, 169, 0x5);                /* Probe Control, trigger on rising edges */

   set_idn(AXIS, 305, 409);                /* Allocate Probe 1 status to Real Time Status */
   set_idn(AXIS, 307, 411);                /* Allocate Probe 2 status to Real Time Status */

   set_idn(AXIS, 170, 0x3);                /* Probe set/enable procedure */

   printf("\nSERCOS drive ready for position latching.\n");

   while (!kbhit())
   {   error(get_drive_status(AXIS, &drive_status));/* Drive Status */
       error(get_position(AXIS, &act));
       drive_status &= (RT_STATUS_1 | RT_STATUS_2);
       printf("\rAct:%10.0lf Status:%x", act, drive_status);
   }
   getch();

   error(get_idn(AXIS, 130, &val));
   printf("\nLatch P1:%ld", val);
   error(get_idn(AXIS, 132, &val));
   printf("\nLatch P2:%ld", val);

   return 0;
}
```

settle3.c

Determine when actual motion has settled

Motion Settling

```
/* SETTLE3.C

:Determine when the actual motion has settled.

When motion is commanded to an axis, the DSP generates a real-time command position trajectory.
Every sample, the DSP calculates an axis' output (analog voltage or pulse rate) based on a PID
servo control algorithm.  The input to the PID control algorithm is the position error.  The posi-
tion error equals the difference between the command position and the actual position (feedback).

The commanded motion is complete when motion_done(...) returns non-zero. Determining the comple-
tion of the actual motion is more difficult.  Since the DSP is always executing a PID algorithm to
control the closed loop system, the motion of the motor is based on the system's response.

This sample shows how to determine the completion of the motion based on the feedback device.  The
motor is considered "settled" when the following criteria are met:

 1) The position error is less than a specified value for 'n' consecutive reads
 2) The rate of change of position error is zero for 'n' consecutive reads

The rate of change of position error is calculated by the DSP as the difference between the posi-
tion error in the current sample, and the position error in the previous sample.  The default sam-
ple rate is 1250 samples per second.

Both the rate of change of position error and the position error are read in the same sample
period using the transfer block.  The transfer block is a special DSP feature that copies a block
of internal memory (up to 64 words) to external memory.  Then the host CPU can read the informa-
tion from external memory.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <conio.h>
# include <stdlib.h>
# include <dos.h>
# include "pcdsp.h"
# include "idsp.h"

# define AXIS               0
# define DISTANCE           20000.0
# define VELOCITY           100000.0
# define ACCELERATION       5000000.0

# define ERROR_WINDOW       5       /* position error in counts */
# define SETTLE_COUNTS      4       /* number of consecutive reads to define settled */
```

```
# define BUFFER_SIZE        1000

int16
   p_err[BUFFER_SIZE],       /* position error from DSP */
   vel[BUFFER_SIZE];         /* actual velocity from DSP */


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;
   }
}

int16 get_settle(int16 axis, int16 * error, int16 * actvel)
{
   P_DSP_DM addr = dspPtr->data_struct + DS_D(0) + DS(axis);
   DSP_DM buffer[2];

   pcdsp_transfer_block(dspPtr, TRUE, FALSE, addr, 2, buffer);

   * error = buffer[0];
   * actvel = buffer[1];

   return dsp_error;
}
```

settle3.c

Determine when actual motion has settled

```
int16 main()
{
   int16
       i, error_code,
       settle_counter = 0,
       data_point = 0;
       int16 start, end;
   double settle_time;

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

   start_r_move(AXIS, DISTANCE, VELOCITY, ACCELERATION);
   while (!motion_done(AXIS))       /* wait for the commanded motion to complete */
       ;

   start = get_dsp_time();          /* read the dsp sample clock */

   while (settle_counter < SETTLE_COUNTS)
   {
       get_settle(AXIS, p_err + data_point, vel + data_point);

       if (!vel[data_point] && (p_err[data_point] < ERROR_WINDOW) &&
              (p_err[data_point] > (-ERROR_WINDOW)))
           settle_counter++;
       else
           settle_counter = 0;

       if (data_point < BUFFER_SIZE)
           data_point++;
   }
   end = get_dsp_time();            /* read the dsp sample clock */

   /* Display the response of the system. */
   for (i = 0; i < data_point; i++)
       printf("\n%4d Vel: %4d Err: %4d", i, vel[i], p_err[i]);

   settle_time = (end - start) * 1000.0 / dsp_sample_rate();
   printf("\nSettling time (msec): %6.4lf", settle_time);

    return 0;
}
```

shome1.c

SERCOS: Homing routine

SERCOS

```
/* SHOME1.C

:SERCOS sample homing routine.

This sample demonstrates a basic homing algorithm.  The home procedure (IDN 148) is activated in
the servo drive.  The drive accelerates (IDN 42) the axis to constant velocity (IDN 41) with
internal position control.  When the home switch is activated, the drive decelerates the axis to
a stop and sets a bit in the "Position Feedback Status" (IDN 403).

Then the command position (IDN 47) is read from the drive and the controller's command/actual
position registers are set with set_position(...).  Finally, the drive procedure is ended with
cancel_exec_procedure(...).

Be sure to initialize the SERCOS communication ring with serc_reset(...) before starting a drive
controlled home procedure.

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 Motion Library Version 2.5
*/


#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "pcdsp.h"
#include "sercos.h"

#define AXIS         0
#define HOME_OFFSET  0L

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;
   }
}
```

shome1.c

```
int main()
{  int16 error_code;
   long pos, done = 0;

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

   set_error_limit(AXIS, 0.0, NO_EVENT);           /* disable controller error limit */
   enable_amplifier(AXIS);                         /* enable drive */

   error(set_idn(AXIS, 52, HOME_OFFSET));          /* home reference distance */
   error(start_exec_procedure(AXIS, 148));         /* start home procedure */

   printf("\nMoving towards home switch (hit any key to quit)");
   while((!done) && (!kbhit()))
       error(get_idn(AXIS, 403, &done));           /* position reference status */

   if (done)
       printf("\nHome found");

   error(get_idn(AXIS, 47, &pos));                 /* command position */
   set_position(AXIS, (double)pos);
   error(cancel_exec_procedure(AXIS, 148));        /* end the home procedure */

   set_error_limit(AXIS, 32767.0, ABORT_EVENT);

   return 0;
}
```

sidn.c

SERCOS: Decode IDN's value(s) based on its attributes

SERCOS

```
/* SIDN.C

:SERCOS sample to decode an IDN's value(s) based on its attributes.

This sample demonstrates how to read the values for an IDN.  Each IDN contains 7 elements (defined
in the SERCOS specification):

Element 1:
 "ID Number"
 size is 16 bits, binary
 mandatory

Element 2:
 "Name of operation data"
 maximum size is 64 bytes
 upper 4 bytes specify the string size
 lower 60 bytes are the string
 optional

Element 3:
 "Attribute of operation data"
 size is 4 bytes, binary
 bits 0-15 represent the conversion factor
 bits 16-18 represent the data length
 bit 19 represents the function of operation data
 bits 20-22 represent the data type and display format
 bits 24-27 represent places after the decimal point
 bits 28-31 reserved
 mandatory

Element 4:
 "Operation data unit"
 maximum size is 16 bytes, binary/string
 bytes 1-2 specify length of programmed text in drive
 bytes 3-4 indicate maximum text length (in bytes) in the drive
 bytes 4-12 variable length character text string
 optional

Element 5:
 "Minimum input value of operation data"
 size is the same as Element 7 (Operation Data), binary
 optional

Element 6:
 "Maximum input value of operation data"
 size is the same as Element 7 (Operation Data), binary
 optional

Element 7:
 "Operation data"
 maximum size is 65532 bytes, binary and/or string
 Fixed length:
   bytes 1-2, operation data, 2 bytes
   bytes 1-4, operation data, 4 bytes
 Variable length:
   bytes 1-2, specify length of programmed data (bytes) in drive
   bytes 3-4, indicate maximum data length available (bytes) in drive
   bytes 5-length, operation data
```

```
For more detailed information regarding the IDN elements, please consult
 the SERCOS specification.

Be sure to initialize the SERCOS communication ring with serc_reset(...)
 before reading an IDN.

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 Motion Library Version 2.5
*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include "idsp.h"
#include "sercos.h"

#ifdef MEI_MSVC20                   /* support for Microsoft Visual C/C++ ver 2.0 */
#  include "medexp.h"               /* prototypes for access to DLL's internal data */
#endif

//#ifdef MEI_MSVC40                 /* support for Microsoft Visual C/C++ ver 4.0 */
#  include "medexp.h"               /* prototypes for access to DLL's internal data */
//#endif

#define BUFFER_SIZE  1024


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;
   }
}

int16 print_list(int16 axis, unsigned16 idn)
{  int16 i, len, p, a=0;
   unsigned16 *buffer, channel, str[MAX_E2_INTS];

   if(get_sercos_channel(axis, &channel))
       return dsp_error;

   buffer = (unsigned16*) calloc(BUFFER_SIZE, sizeof(unsigned16));
   if(read_idn(channel, idn, &len, buffer, TRUE))
       return dsp_error;

   printf("IDN-List for axis %d, IDN %d\n", axis, idn);
   for(i = 0; i < len; i++)
   {   if(get_element_2(channel, buffer[i], str))
           memcpy(str, "No text available", 20);
       p = buffer[i] & 0x8000;
       if(p)
           buffer[i] -= (int16)0x8000;
       printf("%s-%1d-%4d\t%s\n",p?"P":"S", a, buffer[i], str);
   }
   return DSP_OK;
```

```
}

int16 print_long_list(int16 axis, unsigned16 idn)
{  int16 i, len;
   unsigned16 channel;
   unsigned32 *buffer;

   if(get_sercos_channel(axis, &channel))
       return dsp_error;

   buffer = (unsigned32*) calloc(BUFFER_SIZE, sizeof(unsigned32));
   if(read_idn(channel, idn, &len, (unsigned16*)buffer, TRUE))
       return dsp_error;

   for(i = 0; i < len/2; i++)
       printf("%8ld\n",buffer[i]);

   return DSP_OK;
}

int16 print_data_type(IDN_ATTRIBUTES * attr)
{  long proc, type;

   proc = (attr->elem_3 & 0x80000);         /* bit 19 indicates IDN is a Procedure */
   printf("Data function: %s\n", proc?"Procedure Command":"Operation Data or Parameter");
   type = (attr->elem_3>>20) & 0x7L;        /* bit 20-22 represent data type */

   switch (type)
   {
       case 0L:
           printf("Data type: binary\n");
           break;

       case 1L:
       case 3L:
           printf("Data type: unsigned integer\n");
           break;

       case 2L:
           printf("Data type: signed integer\n");
           break;

       case 4L:
           printf("Data type: text\n");
           break;

       case 5L:
           printf("Data type: ID Number (IDN)\n");
           break;

       default:
           printf("Data type: undefined\n");
   }
   return DSP_OK;
}
```

sidn.c

SERCOS: Decode IDN's value(s) based on its attributes

```
int16 print_element_data(int16 axis, unsigned16 idn, int16 size, IDN_ATTRIBUTES * attr)
{  long lval, exp;
   unsigned16 channel, dr_addr;
   char str[200];

   switch (size)
   {
       case 1:       /* fixed data length (2 or 4 bytes) */
       case 2:
           printf("Fixed data length is: %d bytes\n", (size * 2));
           print_data_type(attr);
           exp = (attr->elem_3>>24) & 0xF;          /* bits 24-27 represent decimal places */
           printf("Conversion Factor: %ldx10e-%ld\n", (attr->elem_3 & 0xFFFF), exp);
           printf("Units: %s\n", attr->elem_4);
           printf("Minimum Value: %ld\n", attr->elem_5);
           printf("Maximum Value: %ld\n", attr->elem_6);
           while(!kbhit())
           {  error(get_idn(axis, idn, &lval));
              printf("Value: %10ld (0x%lX)        \r", lval, lval);
           }
           getch();
           break;

       case 4:       /* String */
           printf("Variable data length with 1-byte data strings.\n");
           if(get_sercos_channel(axis, &channel))
              return dsp_error;
           dr_addr = dspPtr->sercdata[channel].drive_addr;
           if(get_idn_string(axis, dr_addr, idn, str))
              return dsp_error;
           printf("String: %s\n", str);
           break;

       case 5:       /* IDN String */
           printf("Variable data length with 2-byte data strings.\n");
           error(print_list(axis, idn));
           break;

       case 6:       /* String (long) */
           printf("Variable data length with 4-byte data strings.\n");
           error(print_long_list(axis, idn));
           break;

       default:
           printf("Data length is undefined\n");
   }
   return DSP_OK;
}
```

```
int main(int argc, char* argv[])
{  int16 axis, error_code, size = 0;
   unsigned16 idn;
   char *endp;
   IDN_ATTRIBUTES attr;

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

   if(argc < 3)
   {   printf("USAGE: SIDN [axis] [idn]");
       return 1;
   }
   axis = (int16) strtol(argv[1], &endp, 10);
   if(!strnicmp("0x", argv[2], 2))
       idn = (unsigned16) strtol(argv[2], &endp, 16);
   else
       idn = (unsigned16) strtol(argv[2], &endp, 10);

   printf("IDN: %u (0x%X)\n", idn, idn);
   error(get_idn_attributes(axis, idn, &attr));    /* read the elements */

   /* Element 1 specifies the ID number.  "P" indicates product specific,
       "S" indicates standard. */
   printf("%s-%d-%d\n", attr.elem_1 & 0x8000 ? "P":"S",
       (attr.elem_1 & 0x7000) >> 12, attr.elem_1 & 0xFFF);
   printf("Name: %s\n", attr.elem_2);              /* Name of operation data */

   size = ((attr.elem_3 >> 16) & 0x7);             /* bits 16-18 represent data length */
   error(print_element_data(axis, idn, size, &attr));

   return DSP_OK;
}
```

sidn1.c

SERCOS: Read IDN value/string

SERCOS

```
/* SIDN1.C

:SERCOS IDN value/string read.

This sample program demonstrates how to read an IDN value or IDN string from a SERCOS node.

Be sure to initialize the SERCOS ring with serc_reset(...) before reading/writing to an IDN.

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 Motion Library Version 2.5
*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "idsp.h"
#include "sercos.h"
#include "sercrset.h"

#define AXIS     0
#define ADDR     1
#define IDN      30  /* Manufacturer Version (string) */


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(void)
{
   int16 error_code, size;
   int32 value;
   char string[MAX_ERROR_LEN];

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

   error(get_idn_size(AXIS, IDN, &size));   /* determine data size */

   if (size)
   {   error(get_idn(AXIS, IDN, &value));
       printf("\nData length(words) is: %d\n", size);
       printf("\nValue: %ld", value);
   }
   else /* variable length string */
   {   error(get_idn_string(AXIS, 1, IDN, string));
       printf("\nData length is variable\n");
       printf("\n%s", string);
   }

   return 0;
}
```

sidnl.c

SERCOS: Read "IDN-List" from IDN in drive

SERCOS


```
/* SIDNL.C

:SERCOS sample to read an "IDN-List" from an IDN in a specific drive.

The "Operation Data" for several IDNs are actually a variable length list of IDNs.  This sample
code demonstrates how to read an "IDN-List" from an IDN.

Here are some useful IDNs (with IDN-Lists):

IDN    Description

17     List of all operation data.
25     List of all procedure commands.
192    List of backup operation data.
187    List of configurable data in the AT
188    List of configurable data in the MDT
21     List of invalid operation data for CP2
22     List of invalid operation data for CP3
23     List of invalid operation data for CP4
18     List of operation data for CP2
19     List of operation data for CP3
20     List of operation data for CP4

Be sure to initialize the SERCOS communication ring with serc_reset(...) before reading the list
of supported IDNs.

Please consult the drive manufacturer's documentation or SERCOS specification for more informa-
tion concerning "IDN-List".

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 Motion Library Version 2.5
*/


#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include "idsp.h"
#include "sercos.h"

#ifdef MEI_MSVC20           /* support for Microsoft Visual C/C++ ver 2.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif

#ifdef MEI_MSVC40           /* support for Microsoft Visual C/C++ ver 4.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif

#define BUFFER_SIZE  1024
#define AXIS         0
#define IDN          17
```

```
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()
{
   int16 error_code, len, value, p, a=0;
   unsigned16 i, *buffer, channel, str[MAX_E2_INTS];

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

   buffer = (unsigned16*) calloc(BUFFER_SIZE, sizeof(unsigned16));

   error(get_sercos_channel(AXIS, &channel));               /* read channel assigned to axis */
   error(read_idn(channel, IDN, &len, buffer, TRUE));       /* read list of idns */

   printf("List of IDN Values for axis: %d\n", AXIS);

   for(i = 0; i < len; i++)
   {   if(get_element_2(channel, buffer[i], str))
           memcpy(str, "No text available", 20);
       p = buffer[i] & 0x8000;
       if(p)
           buffer[i] -= (int16)0x8000;
       printf("%s-%1d-%4d   %s\n",p?"P":"S", a, buffer[i], str);
   }

    return 0;
}
```

sidns.c

SERCOS: Read/write/copy a group of IDNs

SERCOS

```
/* SIDNS.C

:Read/Write/Copy a group of SERCOS IDNs.

This sample code demonstrates how to read/write a list of sequential SERCOS IDN data values.

Be sure to initialize the SERCOS ring with serc_reset(...) before reading/writing to the IDNs.

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 Motion Library Version 2.5
*/


#include <stdio.h>
#include <stdlib.h>
#include "pcdsp.h"
#include "sercos.h"

#define SOURCE_AXIS  0
#define SOURCE_ADDR  1      /* SERCOS Node Address */

#define TARGET_AXIS  1
#define TARGET_ADDR  2      /* SERCOS Node Address */

#define   FIRST_S_IDN       30      /* first Standard IDN */
#define   FIRST_P_IDN       0       /* first Product Specific IDN */

#define   S_IDNS            200
#define   P_IDNS            100

IDNS s_buffer[S_IDNS];
IDNS p_buffer[P_IDNS];


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()
{
   int16 error_code, i;

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

   printf("\nReading Standard IDNs...");
   error(get_idns(SOURCE_AXIS, SOURCE_ADDR, FIRST_S_IDN, S_IDNS, s_buffer));
   printf("done.\n");
   for(i = 0; i < S_IDNS; i++)
       if (!s_buffer[i].error)
           printf("S-0-%4d:\t%ld\n", s_buffer[i].idn, s_buffer[i].value);

   printf("\nReading Drive Specific IDNs...");
   error(get_idns(SOURCE_AXIS, SOURCE_ADDR, (0x8000 + FIRST_P_IDN), P_IDNS, p_buffer));
   printf("done.\n");
   for(i = 0; i < P_IDNS; i++)
       if (!p_buffer[i].error)
           printf("P-0-%4d:\t%ld\n", (p_buffer[i].idn - 0x8000), p_buffer[i].value);

   printf("\nWriting Standard IDNs...");
   error(set_idns(TARGET_AXIS, TARGET_ADDR, S_IDNS, s_buffer));
   printf("done.\n");
   for(i = 0; i < S_IDNS; i++)
       if (!s_buffer[i].error)
           printf("P-0-%4d:\t%ld\n", s_buffer[i].idn, s_buffer[i].value);

   printf("\nWriting Drive Specific IDNs...");
   error(set_idns(TARGET_AXIS, TARGET_ADDR, P_IDNS, p_buffer));
   printf("done.\n");
   for(i = 0; i < P_IDNS; i++)
       if (!p_buffer[i].error)
           printf("P-0-%4d:\t%ld\n", (p_buffer[i].idn - 0x8000), p_buffer[i].value);

   return 0;
}
```

sinit1.c

SERCOS: Initialize ring with Indramat drive

Table 2-1 Functions used to debug ring initialization

Function	*Description*
print_drive_assignment(…)	displays the axis and address assignments for all of the possible axes
print_log(…)	displays any drive messages
print_phase2idns(…)	displays any phase 2 IDNs that were set improperly and are required for loop initialization
print_phase3idns(…)	displays any phase 3 IDNs that were set improperly and are required for loop initialization

SERCOS

```
/* SINIT1.C

:SERCOS initialization with Indramat drive.

This sample program demonstrates SERCOS ring initialization with an Indramat servo drive.

The following functions can be used for debugging the ring initialization:

 print_drive_assignment(...) - displays the axis and address assignments for
 all of the possible axes.

 print_log(...) - displays any drive messages.

 print_phase2idns(...) - displays any phase 2 IDNs that were set improperly
 and are required for loop initialization.

 print_phase3idns(...) - displays any phase 3 IDNs that were set improperly
 and are required for loop initialization.

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 Motion Library Version 2.5
*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "idsp.h"
#include "sercos.h"
#include "sercrset.h"

#ifdef MEI_MSVC20           /* support for Microsoft Visual C/C++ ver 2.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif

#ifdef MEI_MSVC40           /* support for Microsoft Visual C/C++ ver 4.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif
```

```
#define   NODES      1       /* Number of nodes on Sercos ring */
#define   AXIS       0
#define   NODE_ADDR  1

/* Drive Configuration Data */
DRIVE_INFO drive_info[NODES] = {
/* {Axis, Drive Address, Operation Mode, Manufacturer} */
   {AXIS, NODE_ADDR, POSMODE, INDRAMAT}
};


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;
   }
}

void print_drive_assignment(void)
{  int16 i;
   for(i = 0; i < PCDSP_MAX_AXES; i++)
   {   if(assignedDrive[i] > 0)
           printf("Axis %d assigned to Drive %d.\n", i, assignedDrive[i]);
       else
       {   if(assignedDrive[i] == -1)
               printf("Axis %d Drive could not be found.\n", i);
           else
               printf("Axis %d not assigned to any drive.\n", i);
       }
   }
   printf("\n");
}

void print_log(void)
{  int16 i, j;

   for(i = 0; i < dspPtr->axes; i++)
   {   for(j = 0; j < logCount[i]; j++)
           printf("Axis: %d -- %s\n", i, driveMsgs[i][j].msgstr);
   }
}

void print_phase2idns(void)
{  int16 axis, j;

   for(axis = 0; axis < PCDSP_MAX_AXES; axis++)
   {   if(phase2_idncount[axis])
       {   printf("Phase 2 IDNs not set for axis %d:\n", axis);
           for(j = 0; j < phase2_idncount[axis]; j++)
               printf("%d (0x%x)\n", phase2_idnlist[axis][j], phase2_idnlist[axis][j]);
           printf("\n");
       }
   }
}
```

sinit1.c

```
void print_phase3idns(void)
{  int16 axis, j;

   for(axis = 0; axis < PCDSP_MAX_AXES; axis++)
   {   if(phase3_idncount[axis])
       {   printf("Phase 3 IDNs not set for axis %d:\n", axis);
           for(j = 0; j < phase3_idncount[axis]; j++)
              printf("%d (0x%x)\n", phase3_idnlist[axis][j], phase3_idnlist[axis][j]);
           printf("\n");
       }
   }
}

int main(void)
{
   int16 error_code;

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

   error_code = serc_reset(BIT_RATE2, NODES, drive_info);

   print_drive_assignment();
   print_log();

   if(error_code == DSP_SERCOS_DRIVE_INIT)
   {   int16 drives;
       unsigned16 addrs[8], i;
       get_drive_addresses(BIT_RATE2, &drives, addrs);
       for(i = 0; i < drives; i++)
           printf("Drive found at address %d\n", addrs[i]);
   }
   if(error_code == DSP_SERCOS_127_FAILURE)
       print_phase2idns();
   if(error_code == DSP_SERCOS_128_FAILURE)
       print_phase3idns();
   error(error_code);

   enable_amplifier(AXIS);          /* enable servo drive */

   return 0;
}
```

sinit2.c

SERCOS: Initialize ring with Indramat drive (Phase 2, 3)

Table 2-2 Functions used to debug ring initialization with Indramat drive

Function	*Description*
print_drive_assignment(…)	displays the axis and address assignments for all of the possible axes.
print_log(…)	displays any drive messages.
print_phase2idns(…)	displays any phase 2 IDNs that were set improperly and are required for loop initialization.
print_phase3idns(…)	displays any phase 3 IDNs that were set improperly and are required for loop initialization.

SERCOS

```
/* SINIT2.C

:SERCOS initialization with Indramat drive, phase 2,3 IDNs.

This sample program demonstrates SERCOS ring initialization with an Indramat servo drive.  Also,
phase 2 and 3 specific IDNs are configured for initialization with serc_reset(...).

The following functions can be used for debugging the ring initialization:

 print_drive_assignment(...) - displays the axis and address assignments for
 all of the possible axes.

 print_log(...) - displays any drive messages.

 print_phase2idns(...) - displays any phase 2 IDNs that were set improperly
 and are required for loop initialization.

 print_phase3idns(...) - displays any phase 3 IDNs that were set improperly
 and are required for loop initialization.

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 Motion Library Version 2.5
*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "idsp.h"
#include "sercos.h"
#include "sercrset.h"

#ifdef MEI_MSVC20           /* support for Microsoft Visual C/C++ ver 2.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif

#ifdef MEI_MSVC40           /* support for Microsoft Visual C/C++ ver 4.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif
```

```
#define   NODES      1      /* Number of nodes on Sercos ring */
#define   AXIS       0
#define   NODE_ADDR  1

#define PHASE2_IDNS  2      /* Number of Phase 2 IDNs */
#define PHASE3_IDNS  1      /* Number of Phase 3 IDNs */

/* Drive Configuration Data */
DRIVE_INFO drive_info[NODES] = {
/* {Axis, Drive Address, Operation Mode, Manufacturer} */
   {AXIS, NODE_ADDR, POSMODE, INDRAMAT}
};

/* Phase 2 IDNs Configuration Data */
DRIVE_IDNS phase_2_idns[PHASE2_IDNS] = {
/* {IDN, Value, Drive Address} */
   {79, 36000, NODE_ADDR},                 /* Rotational Position Resolution */
   {55, 0x0, NODE_ADDR}             /* Position Polarity */
};

/* Phase 3 IDNs Configuration Data */
DRIVE_IDNS phase_3_idns[PHASE2_IDNS] = {
/* {IDN, Value, Drive Address} */
   {104, 2000, NODE_ADDR}           /* Position Loop Gain */
};


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;
   }
}

void print_drive_assignment(void)
{  int16 i;
   for(i = 0; i < PCDSP_MAX_AXES; i++)
   {   if(assignedDrive[i] > 0)
           printf("Axis %d assigned to Drive %d.\n", i, assignedDrive[i]);
       else
       {   if(assignedDrive[i] == -1)
               printf("Axis %d Drive could not be found.\n", i);
           else
               printf("Axis %d not assigned to any drive.\n", i);
       }
   }
   printf("\n");
}

void print_log(void)
{  int16 i, j;

   for(i = 0; i < dspPtr->axes; i++)
   {   for(j = 0; j < logCount[i]; j++)
           printf("Axis: %d -- %s\n", i, driveMsgs[i][j].msgstr);
   }
}
```

```
void print_phase2idns(void)
{  int16 axis, j;

   for(axis = 0; axis < PCDSP_MAX_AXES; axis++)
   {   if(phase2_idncount[axis])
       {   printf("Phase 2 IDNs not set for axis %d:\n", axis);
           for(j = 0; j < phase2_idncount[axis]; j++)
              printf("%d (0x%x)\n", phase2_idnlist[axis][j], phase2_idnlist[axis][j]);
           printf("\n");
       }
   }
}

void print_phase3idns(void)
{  int16 axis, j;

   for(axis = 0; axis < PCDSP_MAX_AXES; axis++)
   {   if(phase3_idncount[axis])
       {   printf("Phase 3 IDNs not set for axis %d:\n", axis);
           for(j = 0; j < phase3_idncount[axis]; j++)
              printf("%d (0x%x)\n", phase3_idnlist[axis][j], phase3_idnlist[axis][j]);
           printf("\n");
       }
   }
}

int main(void)
{
   int16 error_code;

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

   error(configure_phase2_idns(PHASE2_IDNS, phase_2_idns));
   error(configure_phase3_idns(PHASE3_IDNS, phase_3_idns));
   error_code = serc_reset(BIT_RATE2, NODES, drive_info);

   print_drive_assignment();
   print_log();

   if(error_code == DSP_SERCOS_DRIVE_INIT)
   {   int16 drives;
       unsigned16 addrs[8], i;
       get_drive_addresses(BIT_RATE2, &drives, addrs);
       for(i = 0; i < drives; i++)
           printf("Drive found at address %d\n", addrs[i]);
   }
   if(error_code == DSP_SERCOS_127_FAILURE)
       print_phase2idns();
   if(error_code == DSP_SERCOS_128_FAILURE)
       print_phase3idns();
   error(error_code);

   enable_amplifier(AXIS);   /* enable servo drive */

   return 0;
}
```

sinit3.c

SERCOS: Initialize ring with Indramat drive (user cyclic data)

Table 2-3 Functions used to debug ring initialization with Indramat drive, user-specified cyclic data

Function	*Description*
print_drive_assignment(…)	displays the axis and address assignments for all of the possible axes.
print_log(…)	displays any drive messages.
print_phase2idns(…)	displays any phase 2 IDNs that were set improperly and are required for loop initialization.
print_phase3idns(…)	displays any phase 3 IDNs that were set improperly and are required for loop initialization.

SERCOS

```
/* SINIT3.C

:SERCOS initialization with Indramat drive, user-specified cyclic data.

This sample program demonstrates SERCOS ring initialization with an Indramat servo drive.  Also,
specific IDNs are configured to be sent in the Amplifier Telegram and the Master Data Telegram
(cyclic data).

The following functions can be used for debugging the ring initialization:

 print_drive_assignment(...) - displays the axis and address assignments for
 all of the possible axes.

 print_log(...) - displays any drive messages.

 print_phase2idns(...) - displays any phase 2 IDNs that were set improperly
 and are required for loop initialization.

 print_phase3idns(...) - displays any phase 3 IDNs that were set improperly
 and are required for loop initialization.

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 Motion Library Version 2.5
*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "idsp.h"
#include "sercos.h"
#include "sercrset.h"

#ifdef MEI_MSVC20           /* support for Microsoft Visual C/C++ ver 2.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif

#ifdef MEI_MSVC40           /* support for Microsoft Visual C/C++ ver 4.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
```

```
#endif

#define NODES        1      /* Number of nodes on Sercos ring */
#define AXIS         0
#define NODE_ADDR    1

#define AT_IDNS      1      /* Number of IDNS for the AT */
#define MDT_IDNS     1      /* Number of IDNS for the MDT */

/* Drive Configuration Data */
DRIVE_INFO drive_info[NODES] = {
/* {Axis, Drive Address, Operation Mode, Manufacturer} */
   {AXIS, NODE_ADDR, POSMODE, INDRAMAT}
};

/* AT Configuration Data */
CYCLIC_DATA at_idns[AT_IDNS] = {
/* {IDN, Drive Address} */
   {130, NODE_ADDR}          /* Probe 1 - Positive Edge Position Value */
};

/* MDT Configuration Data */
CYCLIC_DATA mdt_idns[MDT_IDNS] = {
/* {IDN, Drive Address} */
   {91, NODE_ADDR}           /* Bipolar Velocity Limit */
};


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;
   }
}

void print_drive_assignment(void)
{  int16 i;
   for(i = 0; i < PCDSP_MAX_AXES; i++)
   {   if(assignedDrive[i] > 0)
           printf("Axis %d assigned to Drive %d.\n", i, assignedDrive[i]);
       else
       {   if(assignedDrive[i] == -1)
               printf("Axis %d Drive could not be found.\n", i);
           else
               printf("Axis %d not assigned to any drive.\n", i);
       }
   }
   printf("\n");
}

void print_log(void)
{  int16 i, j;

   for(i = 0; i < dspPtr->axes; i++)
   {   for(j = 0; j < logCount[i]; j++)
           printf("Axis: %d -- %s\n", i, driveMsgs[i][j].msgstr);
   }
}
```

sinit3.c

```
void print_phase2idns(void)
{  int16 axis, j;

   for(axis = 0; axis < PCDSP_MAX_AXES; axis++)
   {   if(phase2_idncount[axis])
       {   printf("Phase 2 IDNs not set for axis %d:\n", axis);
           for(j = 0; j < phase2_idncount[axis]; j++)
              printf("%d (0x%x)\n", phase2_idnlist[axis][j], phase2_idnlist[axis][j]);
           printf("\n");
       }
   }
}

void print_phase3idns(void)
{  int16 axis, j;

   for(axis = 0; axis < PCDSP_MAX_AXES; axis++)
   {   if(phase3_idncount[axis])
       {   printf("Phase 3 IDNs not set for axis %d:\n", axis);
           for(j = 0; j < phase3_idncount[axis]; j++)
              printf("%d (0x%x)\n", phase3_idnlist[axis][j], phase3_idnlist[axis][j]);
           printf("\n");
       }
   }
}

int main(void)
{
   int16 error_code;

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

   error(configure_at_data(AT_IDNS, at_idns));
   error(configure_mdt_data(MDT_IDNS, mdt_idns));
   error_code = serc_reset(BIT_RATE2, NODES, drive_info);

   print_drive_assignment();
   print_log();

   if(error_code == DSP_SERCOS_DRIVE_INIT)
   {   int16 drives;
       unsigned16 addrs[8], i;
       get_drive_addresses(BIT_RATE2, &drives, addrs);
       for(i = 0; i < drives; i++)
           printf("Drive found at address %d\n", addrs[i]);
   }
   if(error_code == DSP_SERCOS_127_FAILURE)
       print_phase2idns();
   if(error_code == DSP_SERCOS_128_FAILURE)
       print_phase3idns();
   error(error_code);

   enable_amplifier(AXIS);          /* enable servo drive */

   return 0;
}
```

SERCOS: Initialize ring with Indramat drive (user cyclic data)

sinit4.c

SERCOS: Initialize loop with Lutze ComCon/16-bit I/O modules

SERCOS

```
/* SINIT4.C

:SERCOS loop initialization with Lutze ComCon, 16bit input and output modules.

This sample program demonstrates SERCOS loop initialization with a Lutze ComCon slave module.
Additionally, a 16bit input and a 16-bit output module are mounted in the Lutze carrier unit
(backplane).

The I/O modules are mapped into the cyclic data and updated by the DSP every SERCOS loop cycle.
The Inputs are mapped into the "AT" and the Outputs are mapped into the "MDT".  The cyclic data
can be accessed from the host with the functions read_cyclic_at_data(...) and read/
write_cyclic_mdt_data(...).


Here are the configurations:

SERCOS Baud Rate = 2Mbit

Lutze ComCon:        Axis = 0,      SERCOS Address = 10
Lutze Input Module:  Address = 2066
Lutze Output Module:  Address = 2086

Note:  The Lutze I/O module addresses (IDNs in cyclic data) are switch selectable. Please consult
the Lutze documentation for more details.

The following functions can be used for debugging the ring initialization:

 print_drive_assignment(...) - displays the axis and address assignments for
 all of the possible axes.

 print_log(...) - displays any drive messages.

 print_phase2idns(...) - displays any phase 2 IDNs that were set improperly
 and are required for loop initialization.

 print_phase3idns(...) - displays any phase 3 IDNs that were set improperly
 and are required for loop initialization.

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 Motion Library Version 2.5
*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "idsp.h"
#include "sercrset.h"

#ifdef MEI_MSVC20           /* support for Microsoft Visual C/C++ ver 2.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif

#ifdef MEI_MSVC40           /* support for Microsoft Visual C/C++ ver 4.0 */
#  include "medexp.h"       /* prototypes for access to DLL's internal data */
#endif
```

```
#define NODES        1      /* Number of nodes on Sercos ring */
#define AXIS         0
#define NODE_ADDR    10

#define AT_IDNS      1      /* Number of IDNs for the AT */
#define MDT_IDNS     1      /* Number of IDNs for the MDT */

/* Drive Configuration Data */
DRIVE_INFO drive_info[NODES] = {
/* {axis, Sercos drive address, drive type, drive manufacturer} */
   {AXIS, NODE_ADDR, USERMAP, LUTZE}
};

/* AT Configuration data */
CYCLIC_DATA at_idns[AT_IDNS] = {
/* {idn, Sercos drive address} */
   {2066, NODE_ADDR}         /* IDN is the address of the Input module */
};

/* MDT Configuration Data */
CYCLIC_DATA mdt_idns[MDT_IDNS] = {
/* {idn, Sercos drive address} */
   {2086, NODE_ADDR}         /* IDN is the address of the Output module */
};


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;
   }
}

void print_drive_assignment(void)
{  int16 i;
   for(i = 0; i < PCDSP_MAX_AXES; i++)
   {   if(assignedDrive[i] > 0)
           printf("Axis %d assigned to Drive %d.\n", i, assignedDrive[i]);
       else
       {   if(assignedDrive[i] == -1)
              printf("Axis %d Drive could not be found.\n", i);
           else
              printf("Axis %d not assigned to any drive.\n", i);
       }
   }
   printf("\n");
}

void print_log(void)
{  int16 i, j;

   for(i = 0; i < dspPtr->axes; i++)
   {   for(j = 0; j < logCount[i]; j++)
           printf("Axis: %d -- %s\n", i, driveMsgs[i][j].msgstr);
   }
}
```

```
void print_phase2idns(void)
{  int16 axis, j;

   for(axis = 0; axis < PCDSP_MAX_AXES; axis++)
   {   if(phase2_idncount[axis])
       {   printf("Phase 2 IDNs not set for axis %d:\n", axis);
           for(j = 0; j < phase2_idncount[axis]; j++)
              printf("%d (0x%x)\n", phase2_idnlist[axis][j], phase2_idnlist[axis][j]);
           printf("\n");
       }
   }
}

void print_phase3idns(void)
{  int16 axis, j;

   for(axis = 0; axis < PCDSP_MAX_AXES; axis++)
   {   if(phase3_idncount[axis])
       {   printf("Phase 3 IDNs not set for axis %d:\n", axis);
           for(j = 0; j < phase3_idncount[axis]; j++)
              printf("%d (0x%x)\n", phase3_idnlist[axis][j], phase3_idnlist[axis][j]);
           printf("\n");
       }
   }
}

int main(void)
{
   int16 error_code, lutze_in, lutze_out;

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

   error(configure_at_data(AT_IDNS, at_idns));              /* input data */
   error(configure_mdt_data(MDT_IDNS, mdt_idns));           /* output data */
   error_code = serc_reset(BIT_RATE2, NODES, drive_info);

   print_drive_assignment();
   print_log();

   if(error_code == DSP_SERCOS_DRIVE_INIT)
   {   int16 drives;
       unsigned16 addrs[8], i;
       get_drive_addresses(BIT_RATE2, &drives, addrs);
       for(i = 0; i < drives; i++)
           printf("Drive found at address %d\n", addrs[i]);
   }
   if(error_code == DSP_SERCOS_127_FAILURE)
       print_phase2idns();
   if(error_code == DSP_SERCOS_128_FAILURE)
       print_phase3idns();
   error(error_code);

   enable_amplifier(AXIS);                                  /* enable Lutze block */
   write_cyclic_mdt_data(AXIS, 0, 0x5555);/* write the outputs */

   printf("Lutze I/O.\n");
   while (!kbhit())
   {   lutze_in = read_cyclic_at_data(AXIS, 0);             /* read inputs */
       lutze_out = read_cyclic_mdt_data(AXIS, 0);           /* read outputs */
       printf("\rIn:0x%4x Out:0x%4x", lutze_in, lutze_out);
   }
   getch();

   return 0;
}
```

slave.c

Command motion with linked axes

Electronic Gearing

```
/* SLAVE.C

:Simple demonstration of commanded motion with linked axes.

This code links 2 axes together by actual position. Then the master axis is put in Idle mode so
its motor shaft can be turned by hand. The DSP will command the slave axis to follow the master.
Then the slave is commanded to move. Then the master is put into Run mode and commanded to move.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"

# define  MASTER 0
# define  SLAVE  1
# define  RATIO  1.0

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;
   }
}
```

```
int16 display(void)
{
   double actual, command;

   while (!kbhit())                 /* wait for a key press */
   {   get_command(SLAVE, &command);
       get_position(SLAVE, &actual);
       printf("Slave Cmd: %8.0lf Act: %8.0lf\r", command, actual);
   }
   getch();

   return 0;
}

int16 main()
{
   int16 error_code;

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

   mei_link(MASTER, SLAVE, RATIO, LINK_ACTUAL);
   controller_idle(MASTER);                         /* disable PID control */
   printf("\nAxes linked, Master is in Idle mode.\n");
   display();

   start_move(SLAVE, 4000.0, 1000.0, 4000.0);       /* move the SLAVE axis */
   printf("\nCommanding slave axis to move\n");
   display();

   controller_run(MASTER);                          /* enable PID control */

   start_move(MASTER, 4000.0, 1000.0, 4000.0);      /* move linked axes */
   start_move(MASTER, -4000.0, 1000.0, 4000.0);
   printf("\nCommanding master axis to move\n");
   display();


   endlink(SLAVE);                                  /* unlink axes */

    return 0;
}
```

snfind.c

SERCOS: Find node addresses

SERCOS

```
/* SNFIND.C

:SERCOS Node find.

This sample program demonstrates how to find the addresses of Nodes connected to a SERCOS ring.

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 Motion Library Version 2.5
*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "idsp.h"
#include "sercos.h"
#include "sercrset.h"


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(void)
{
   int16 error_code, i, nodes;
   unsigned16 addrs[8];

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

   error(get_drive_addresses(BIT_RATE2, &nodes, addrs));
   for(i = 0; i < nodes; i++)
       printf("SERCOS Node found at address %d\n", addrs[i]);

   return 0;
}
```

speed.c

Determine function execution time (how long)

Performance Benchmarking

```
/* SPEED.C

:Test function execution time.

This code uses the DSP's sample clock to calculate the execution time for  several library func-
tions.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


#  include <stdio.h>
#  include <stdlib.h>
#  include <dos.h>
#  include "pcdsp.h"


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;
   }
}

void calculate_time(int16 t1, int16 t2, int16 cycles)
{
   int16 delta, sample_rate;
   double e_time;

   sample_rate = dsp_sample_rate();
    delta = t2 - t1;
    e_time = 1.e6 * (delta)/((double)cycles * (double)sample_rate);
    printf("%d cycles in %d samples (%lf microseconds/cycle).\n\n",
          cycles, delta, e_time);
}
```

speed.c

Determine function execution time (how long)

```
void test_no_function(int16 cycles)
{
   int16 i, t1, t2;

    printf("\nBasic Cycle (no function called)\n");

    disable();
    t1 = dsp_read_dm(0x11F);

    for(i = 0; i < cycles; i++)
      ;

    t2 = dsp_read_dm(0x11F);
    enable();
    calculate_time(t1, t2, cycles);
}

void test_dsp_read_dm(int16 cycles)
{
   int16 i, t1, t2;

    printf("dsp_read_dm() function\n");

    disable();
    t1 = dsp_read_dm(0x11F);

    for(i = 0; i < cycles; i++)
        dsp_read_dm(0x11F);

    t2 = dsp_read_dm(0x11F);
    enable();
    calculate_time(t1, t2, cycles);
}

void test_dsp_write_dm(int16 cycles)
{
   int16 i, t1, t2;

    printf("dsp_write_dm() function\n");

    disable();
    t1 = dsp_read_dm(0x11F);

    for(i = 0; i < cycles; i++)
        dsp_write_dm(0x100,0);

    t2 = dsp_read_dm(0x11F);
    enable();
    calculate_time(t1, t2, cycles);
}

void test_get_io(int16 cycles)
{
   int16 i, t1, t2, value;

    printf("get_io() function\n");

    disable();
    t1 = dsp_read_dm(0x11F);

    for(i = 0; i < cycles; i++)
        get_io(0, &value);

    t2 = dsp_read_dm(0x11F);
    enable();
    calculate_time(t1, t2, cycles);
}
```

```
void test_set_io(int16 cycles)
{
   int16 i, t1, t2;

    printf("set_io() function\n");

    disable();
    t1 = dsp_read_dm(0x11F);

    for(i = 0; i < cycles; i++)
        set_io(0, 0);

    t2 = dsp_read_dm(0x11F);
    enable();
    calculate_time(t1, t2, cycles);
}

void test_get_command(int16 cycles)
{
   int16 i, t1, t2;
   double value;

    printf("get_command() function\n");

    disable();
    t1 = dsp_read_dm(0x11F);

    for(i = 0; i < cycles; i++)
        get_command(0, &value);

    t2 = dsp_read_dm(0x11F);
    enable();
    calculate_time(t1, t2, cycles);
}

void test_set_command(int16 cycles)
{
   int16 i, t1, t2;
   double value = 0.0;

    printf("set_command() function\n");

    disable();
    t1 = dsp_read_dm(0x11F);

    for(i = 0; i < cycles; i++)
    {  set_command(0, value);
        value += 1.0;
   }

    t2 = dsp_read_dm(0x11F);
    enable();
    calculate_time(t1, t2, cycles);
}
```

speed.c

Determine function execution time (how long)

```
void test_get_position(int16 cycles)
{
   int16 i, t1, t2;
   double value;

    printf("get_position() function\n");

    disable();
    t1 = dsp_read_dm(0x11F);

    for(i = 0; i < cycles; i++)
        get_position(0, &value);

    t2 = dsp_read_dm(0x11F);
    enable();
    calculate_time(t1, t2, cycles);
}

void test_set_position(int16 cycles)
{
   int16 i, t1, t2;
   double value = 0.0;

    printf("set_position() function\n");

    disable();
    t1 = dsp_read_dm(0x11F);

    for(i = 0; i < cycles; i++)
    {  set_position(0, value);
        value += 1.0;
    }

    t2 = dsp_read_dm(0x11F);
    enable();
    calculate_time(t1, t2, cycles);
}

void test_get_analog(int16 cycles)
{
   int16 i, t1, t2, value;

    printf("get_analog() function\n");

    disable();
    t1 = dsp_read_dm(0x11F);

    for(i = 0; i < cycles; i++)
      get_analog(0, &value);

    t2 = dsp_read_dm(0x11F);
    enable();
    calculate_time(t1, t2, cycles);
}
```

```
void test_start_r_move(int16 cycles)
{
   int16 i, t1, t2;

    printf("start_r_move() function\n");
    error(dsp_reset());

    disable();
    t1 = dsp_read_dm(0x11F);

    for(i = 0; i < cycles; i++)
      start_r_move(0, 1000.0, 10000.0, 100000.0);

    t2 = dsp_read_dm(0x11F);
    enable();
    calculate_time(t1, t2, cycles);
}

void test_start_move(int16 cycles)
{
   int16 i, t1, t2;
   double value = 0.0;

    printf("start_move() function\n");
    error(dsp_reset());

    disable();
    t1 = dsp_read_dm(0x11F);

    for(i = 0; i < cycles; i++)
    {  start_move(0, value, 10000.0, 100000.0);
        value += 1000.0;
   }

    t2 = dsp_read_dm(0x11F);
    enable();
    calculate_time(t1, t2, cycles);
}

int16 main()
{
    int16 error_code;

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

    printf("\nSpeeds measured with the DSP sample timer (%d Hz)\n", dsp_sample_rate());

   test_no_function(30000);

   test_dsp_read_dm(30000);
   test_dsp_write_dm(30000);

   test_get_io(30000);
   test_set_io(30000);

   test_get_command(500);
   test_set_command(500);

   test_get_position(500);
   test_set_position(500);

   test_get_analog(10000);

   test_start_r_move(100);
   test_start_move(100);
    return 0;
}
```

state.c

Display axis state/status/source info

Debugging Routines

```
/* STATE.C

:Displays axis state, status, and source.

This program is written in the form of a state machine.

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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/

# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include <string.h>
# include "pcdsp.h"

# define  READY             0
# define  MOVE              1
# define  GOTO_0            2
# define  AXIS_COMMAND      3

int16   state;

char * states[] = {"running",
                   "running",
                   "new frame",
                   "","","","","",
                   "stop event",
                   "",
                   "e stop event",
                   "","","",
                   "abort event"};

char * sources[] = {"none",
                    "home switch",
                    "pos limit switch",
                    "neg limit switch",
                    "amp fault",
                    "not available",
                    "not available",
                    "neg position limit",
                    "pos position limit",
                    "error limit",
                    "pc command",
                    "out of frames",
                    "temposonics probe fault",
                    "axis command"};
```

```
char * stati[] = {" ",
                  "in sequence ",
                  "in position ",
                  "in motion ",
                  "neg direction ",
                  "frames left "};


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;
   }
}

void display_axis_state(int16 axis)
{
   int16      state, source, status, i;
   char     state_string[25], source_string[25], status_string[65];

   state = axis_state(axis);
   source = axis_source(axis);
   status = axis_status(axis);

   strcpy(state_string, states[state]);
   strcpy(source_string, sources[source]);

   strcpy(status_string, "");
   for(i = 0; i < 5; i++)
      if(status & (1 << (4 + i)))
         strcat(status_string, stati[i + 1]);
   // Fill the rest of the status buffer with " ".
   for(i = 0; i < (66 - strlen(status_string)); i++)
      strcat(status_string, stati[0]);

   printf("%s\t%s\t%s\r", state_string, source_string, status_string);
}

void CheckState(int16 axis)
{
   switch(state)
   {
      case READY:
         break;

      case MOVE:
         start_move(axis, 4000.0, 1000.0, 10000.0);
         state = READY;
         break;

      case  GOTO_0:
         start_move(axis, 0.0, 1000.0, 10000.0);
         state = READY;
         break;

      case AXIS_COMMAND:
         dsp_axis_command(1, axis, STOP_EVENT);
         state = READY;
         break;
```

state.c

Display axis state/status/source info

```
   }
}

int16 main()
{
   int16   done = FALSE, axis = 0;
   int16 error_code;

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

   state = READY;

   printf("<ESC> = exit, m = move, 0 = goto 0, c = send stop event from another axis\n");
   while(!done)
   {
      CheckState(axis);
      display_axis_state(axis);

      if (kbhit())
      {
         switch(getch())
         {
            case 0x1B:
               done = TRUE;
               break;

            case 'm':
               state = MOVE;
               break;

            case '0':
               state = GOTO_0;
               break;

            case 'c':
               state = AXIS_COMMAND;
               break;

            case READY:
               break;
         }
      }
   }
   return 0;
}
```

stoplat.c

Move, latch positions, generate stop events & back off

Position Latching

```
/* STOPLAT.C

:Move, latch positions, generate stop events, and back off a relative distance.

This sample creates a sequence of frames on a phantom axis.  The frames cause the DSP to latch
positions and generate Stop Events based on an User I/O bit. The steps are:

 1) Configure a phantom axis (requires n+1 axis firmware on an n axis card)
 2) Download frames to trigger Stop Events when User I/O bit #22 goes low
 3) Configure and arm position latching
 4) Command motion on 3 axes
 5) Wait for position latch and Stop Event
 6) Clear the Stop Events
 7) Command a relative move in the opposite direction

Be sure user I/O bit #22 is normally driven high, and is pulled low to activate the latch.  The
falling edge of bit #22 triggers the DSP's interrupt.  The DSP's interrupt routine handles the
latching of the actual positions of all axes within 4 microseconds.

The phantom axis is created by downloading 4axis.abs to a 3-axis board.

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
*/

/*  Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include "pcdsp.h"
# include "idsp.h"

# define AXES        4
# define PHANTOM     3

# define LATCH_BIT   22
# define VELOCITY    1000.0
# define ACCEL       10000.0

# define STOP_RATE   10000.0        /* Stop Event Deceleration */


double
   dist[] = {25000.0, 25000.0, 25000.0},
   back_off_dist[] = {7000.0, 8000.0, 5000.0};

int16 back_dir[PCDSP_MAX_AXES] = {0, 0, 0};
```

stoplat.c

Move, latch positions, generate stop events & back off

```
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;
   }
}

void disable_hardware_limits(int16 axis)
{
   set_positive_limit(axis, NO_EVENT);
   set_negative_limit(axis, NO_EVENT);
   set_home(axis, NO_EVENT);
   set_amp_fault(axis, NO_EVENT);
}

void disable_software_limits(int16 axis)
{
   int16 action;
   double position;

   get_positive_sw_limit(axis, &position, &action);
   set_positive_sw_limit(axis, position, NO_EVENT);
   get_negative_sw_limit(axis, &position, &action);
   set_negative_sw_limit(axis, position, NO_EVENT);
   get_error_limit(axis, &position, &action);
   set_error_limit(axis, position, NO_EVENT);
}

void stop_when_latched(int16 phantom, int16 n_axes, int16 * map)
{
   int16 i;

   /* Download frames to generate a Stop Event when the LATCH_BIT goes low. */
   dsp_io_trigger(phantom, LATCH_BIT, FALSE);
   for (i = 0; i < n_axes; i++)
       dsp_axis_command(phantom, map[i], STOP_EVENT);
}

void display(int16 n_axes, int16 * map)
{
    int16 i;
    double cmd;

    printf("\r");

    for (i = 0; i < n_axes; i++)
    { get_command(map[i], &cmd);
      printf("%d: %10.0lf ", i, cmd);
    }
}
```

```
int16 end_of_motion(int16 n_axes, int16 * map)
{
    int16 i;

    for(i = 0; i < n_axes; i++)
    {  if(!motion_done(map[i]))
            return 0;
    }
    return 1;
}


void recover(int16 n_axes, int16 * map)
{
   int16 i, temp_state;

   while (!end_of_motion(n_axes, map))
       display(n_axes, map);

   for (i = 0; i < n_axes; i++)
   {   error(clear_status(map[i]));
       start_r_move(map[i], (back_dir[i] * back_off_dist[i]), VELOCITY, ACCEL);
       back_dir[i] = 0;              /* reset the direction */
   }
}

void initialize(int16 phantom, int16 n_axes, int16 * map)
{
   int16 i, error_code;

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

   disable_hardware_limits(phantom);        /* prevent unintended events */
   disable_software_limits(phantom);

   init_io(2, IO_INPUT);
   for (i = 0; i < n_axes; i++)
       set_stop_rate(map[i], STOP_RATE);

   stop_when_latched(PHANTOM, n_axes, map);
   arm_latch(TRUE);
}
```

stoplat.c

Move, latch positions, generate stop events & back off

```
int16 main()
{
   int16 i, n_axes, axes[] = {0, 1, 2};
   double position;

   n_axes = (sizeof(axes)/sizeof(int16));

   initialize(PHANTOM, n_axes, axes);
   printf("\nToggle bit #22 to latch positions.\n\n");

   set_gates(n_axes, axes);
   for (i = 0; i < n_axes; i++)
   {   start_r_move(axes[i], dist[i], VELOCITY, ACCEL);
       /* set the back off direction */
       if (dist[i] > 0)
           back_dir[i] = -1;
       if (dist[i] < 0)
           back_dir[i] = 1;
   }
   reset_gates(n_axes, axes);

   while (!latch_status())
       display(n_axes, axes);

   for (i = 0; i < n_axes; i++)
   {   get_latched_position(axes[i], &position);
       printf("\nAxis: %d Latched Pos: %12.0lf", axes[i], position);
   }
   printf("\n");

   recover(n_axes, axes);
   while (!end_of_motion(n_axes, axes))
       display(n_axes, axes);

   return 0;
}
```

stoplat.c

Move, latch positions, generate stop events & back off

template.c

A simple motion control program template

Configuration Files

```
/* TEMPLATE.C

Shows how to access various functions of the MEI controller: simple initialization & communica-
tions.



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
*/

/* Revision Control System Information
   $Source$
   $Revision$
   $Date$

   $Log$
*/


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


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()
{
   int16 error_code;

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

    return 0;
}
```

Numerics

A

B

C

D

E

F

G

H

I

J

L

M

O

P

Q

R

S

T

U

V

W

W

T

L