

Table of Contents

KMotion/KFLOP Executive Software Screens

Bode Plot Screen	Page 3	GCode Screen	Page 12
C Program Screen	Page 14	IIR Filter Screen	Page 18
Configuration/ FLASH Screen	Page 24	Step Response Screen	Page 35
Console Screen	Page 44	Axis Screen	Page 46

KFLOP Board specific

Summary	Page 47	Quick Start/USB Driver Installation	Page 48
HW/SW Specification	Page 55	Board Layout	Page 57
Block Diagram	Page 59	Hardware/Connector Description	Page 60
Analog IO Screen	Page 71	Digital IO Screen	Page 74
Functional Diagram	Page 79		

General

Servo Flow Diagram	Page 80	Driver Library Routines	Page 81
Script Commands	Page 94	Driver Library Flow Diagram	Page 159
Using Multiple Boards	Page 160	Preemptive Multi-tasking	Page 162
RS232/UART	Page 163		

KMotionCNC & Mach3

KMotionCNC	Page 167	Mach3 Plugin	Page 205
----------------------------	-----------------	------------------------------	-----------------

Kanalog

Specification	Page 235	Kanalog Hardware/Connectors	Page 236
BlockDiagram	Page 245	Board Layout	Page 246
Kanalog Use and Settings	Page 248		

SnapAmp

SnapAmp Use and Settings	Page 252	SnapAmp Hardware/Connectors	Page 256
--	-----------------	---	-----------------

Examples

Step/Direction Output Mode	Page 263	Brush Motor/SnapAmp Example	Page 274
Closed Loop Step/Dir t Mode	Page 289	Resolver as User Input Mode	Page 296
Data Gather Example	Page 305		

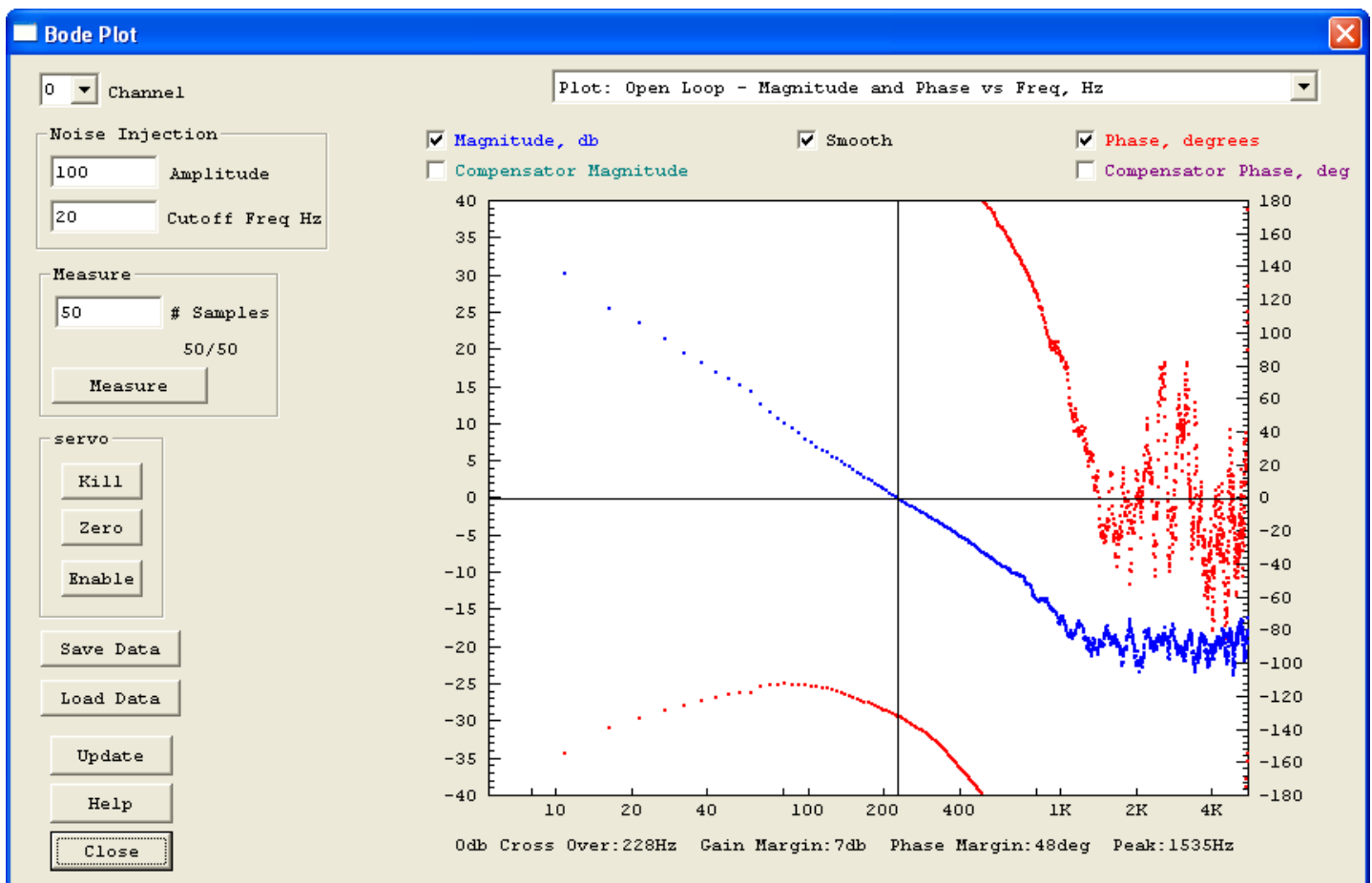
Videos

Step and Direction		Brush Motor with SnapAmp	
Resolver with KMotion		Nonlinear Kinematics	
IR Remote Control		How Parameters can be Set	

Forum/Support

Dynomotion Yahoo Group		Searchable Forum Archive	
--	--	--	--

Bode Plot Screen



(Click on above image to jump to relative topic)

The **Bode Plot Screen** allows the measurement of a servo loop and graphs the open loop response. A Bode Plot is by far the most common means used to measure and understand the behavior of a control loop. **KMotion** contains advanced built-in features to allow rapid Bode Plot measurement and display. The current PID and IIR Filter transfer functions may also be superimposed and graphed.

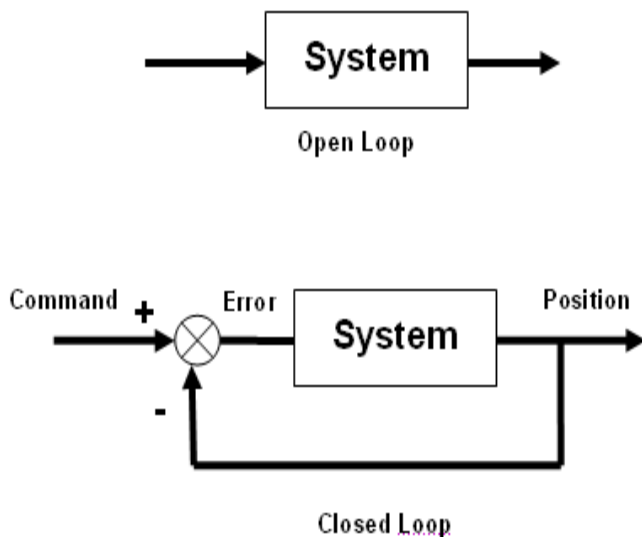
A Bode Plot is a plot of magnitude and phase of a system with respect to frequency. Any linear system that is driven by a sinusoidal input for a long time, will output an sinusoidal signal of the same frequency. The output signal may be shifted in phase and of a different magnitude than the input. A Bode plot is a graph of both the change in phase and the relative change in magnitude (expressed in decibels, db), as a function of frequency.

A Bode plot is a useful tool used to examine the stability of a servo feedback loop. If a system has an open loop gain of -1 (magnitude of 0 db and phase of -180 degrees), then if it is placed into a negative feedback loop, it will become unstable and oscillate. Because a system's gain and phase vary as function of frequency, if the system has a magnitude of 0db and phase of -180 degrees at *any frequency* it will be unstable and oscillate at that frequency. The way to avoid an unstable

system is to avoid having simultaneous conditions of 0db and -180 degrees occur at any frequency. Where the magnitude of the system is 0db the amount that the phase is different from -180 degrees is called the *phase margin*. The larger the phase margin the more stable the system. Similarly where the phase is -180 degrees the amount that the magnitude is different from 0db is called the *gain margin*. Again the larger the gain margin, the more stable the system. As a general rule of thumb, for a system to be reasonably stable it should have a phase margin of at least 30 degrees and a gain margin of at least 3 db.

The Bode Plot Screen attempts to identify and measure the 0 db crossover frequency (the first point where the open loop magnitude becomes less than 1, often defined as the system bandwidth, 228 Hz on the example above), the gain and phase margins, and one or two sharp peaks in the magnitude data after the crossover. Some mechanical systems have sharp resonant peaks that may cause the system to go unstable if these peaks approach the 0 db line and have phase near -180 degrees. A notch filter placed at these frequencies may increase performance. The measurements are displayed under the graph as shown below.

0db Cross Over:228Hz Gain Margin:7db Phase Margin:48deg Peak:1535Hz



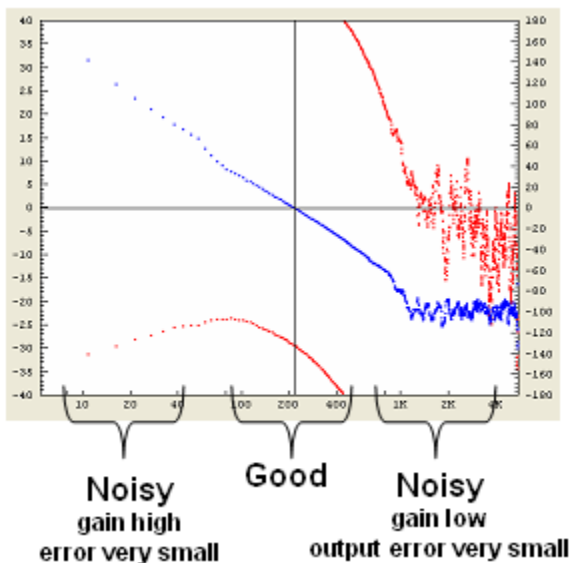
The most direct method for obtaining the open loop response, is to break the servo loop open, inject a signal into the system and measure the output. However this is usually impractical as most systems will run out of their linear range if driven in an open loop manner. **KMotion** operates the servo loop in its normal closed loop form, injects a command signal, measures the position response, and mathematically derives the open loop response. This does require that the servo loop function in some form as a stable closed loop servo *before* a measurement may be made. Performance is not a requirement so

low gains might be used to obtain an initial stable system.

To perform a *Bode Plot* Measurement: select the *channel* to measure, select the desired [Amplitude and Cutoff Frequency](#) for the stimulus to be injected, select the *# of samples* to average, and depress the *Measure* Pushbutton. All current Configuration Parameters (from the [Configuration Screen](#)), Tuning Parameters (from the [Step Response Screen](#)), and Filter Parameters (from the [IIR Filter Screen](#)) will be downloaded, the selected Axis channel will be enabled, and the measurement will begin.

While the measurement is in progress the number of samples acquired will be displayed and the *Measure* Pushbutton will change to a *Stop* Pushbutton. Pushing the Stop button will stop acquiring data after the current sample is complete.

Depending on the type of plot requested (either Time Domain or Frequency Domain) either the last acquired time domain measurement will be displayed or the *average* all the frequency domain measurement so far acquired will be displayed.



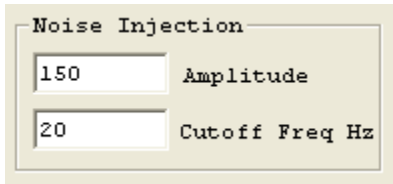
Unfortunately Bode Plots often have regions that are very noisy. But *fortunately* these are almost always in regions that are not important to us. At frequencies where the open loop gain is very high (usually at low frequencies), the servo loop performs very well, and the *Position* closely matches the *Command* signal. In this case, the calculation of the *Error* signal (see above) is calculated by taking the difference between two nearly equal values. A small error in Position measurement will then result in a relatively large error in the calculated error value. Similarly, when the system has a very low gain (usually at high frequencies), the position signal is very small and often highly influenced by noise, or if an encoder is used, by encoder resolution. The regions of interest in order to determine system stability, are where the open loop gain is near 0db and the measurements

are normally very accurate.

Additionally, instead of injecting sine waves at various frequencies individually, **KMotion** uses a technique where a random noise signal that is rich in *many* frequencies is injected. Using a method involving an FFT (Fast Fourier Transform) of the input and output, the entire frequency response may be obtained at once.

Bode Plot analysis is fundamentally based on the assumption that the system being analyzed is *linear*. Linear in this context means that any signal injected into a system that provides a response, might be broken into parts, each piece injected separately, and all the resulting responses when summed would equal the original response. If a system being measured does not meet this criteria

then the result is basically useless and meaningless. Masses, Springs, Dampers, Inductors, Resistors, Capacitors, and all combinations thereof are examples of devices which produce very linear effects. Static friction, Saturation, and Encoder count quantization, are examples of non-linear effects.

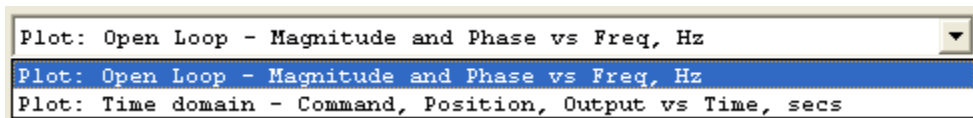
A control panel titled "Noise Injection" with two input fields. The first field contains the value "150" and is labeled "Amplitude". The second field contains the value "20" and is labeled "Cutoff Freq Hz".

Noise Injection	
150	Amplitude
20	Cutoff Freq Hz

It is therefore very important to verify that while the system is being measured that it is operating in its linear range. This usually entails that the system isn't being driven too hard (or too fast), so that the drive to the system (*Output*) is not reaching saturation. Additionally, it is important to verify that the system is being driven sufficiently hard (or slowly enough) that a measurable *Position* change is being

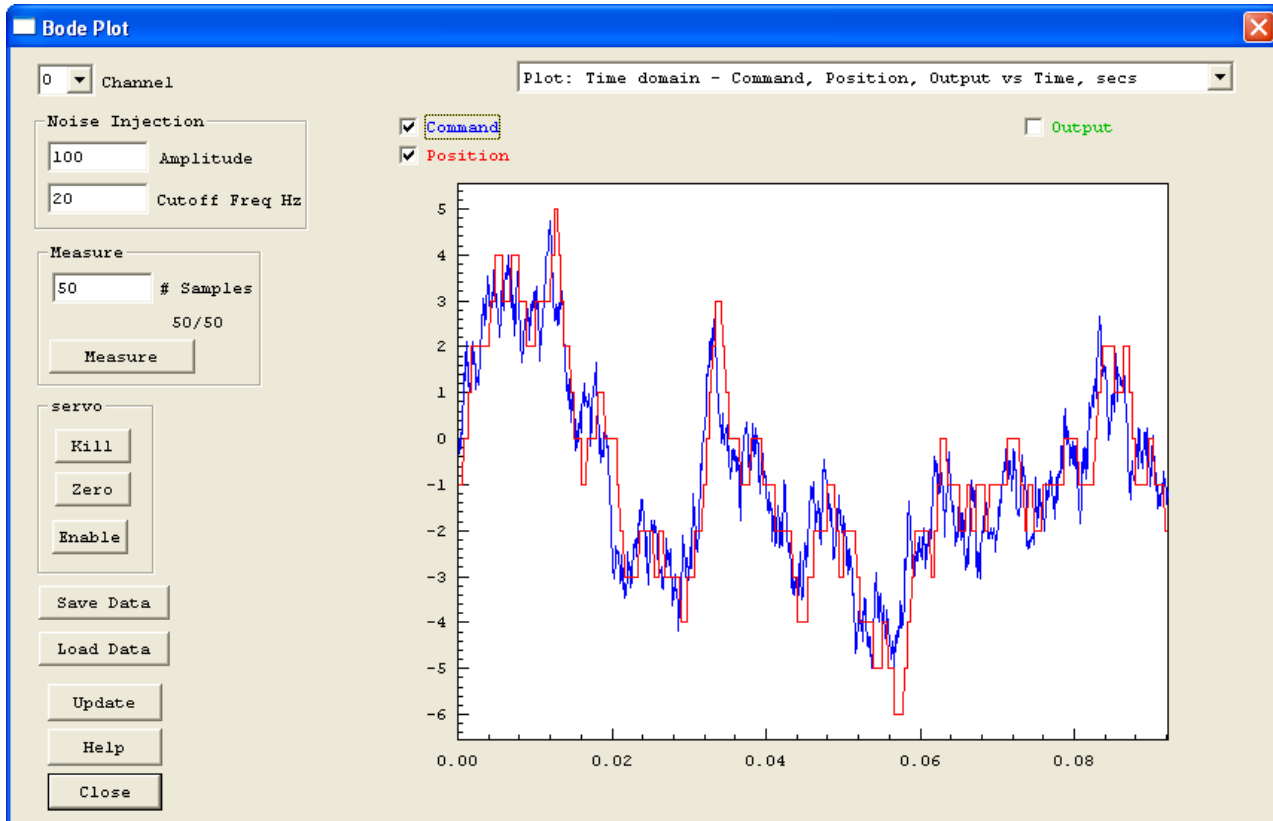
observed. The *Noise Injection Amplitude* and *Cutoff Frequency* should be adjusted to optimize these conditions. *Noise Amplitude* has the same units as *Position* Measurement. It should be noted that setting the *Cutoff Frequency* very low, may reduce the high frequency stimulation to the system to such a point that the higher frequency measurements are invalid or very noisy.

The Bode Plot Screen allows the measurement data to be viewed in the *time domain* in order to check the signal amplitudes so that the optimal signal levels may be used in order to minimize the non-linear effects of saturation and quantization. Select the Plot Type drop down list shown below to switch between frequency domain and time domain displays.

A dropdown menu with three options. The first option is "Plot: Open Loop - Magnitude and Phase vs Freq, Hz". The second option is "Plot: Open Loop - Magnitude and Phase vs Freq, Hz" and is highlighted in blue. The third option is "Plot: Time domain - Command, Position, Output vs Time, secs".

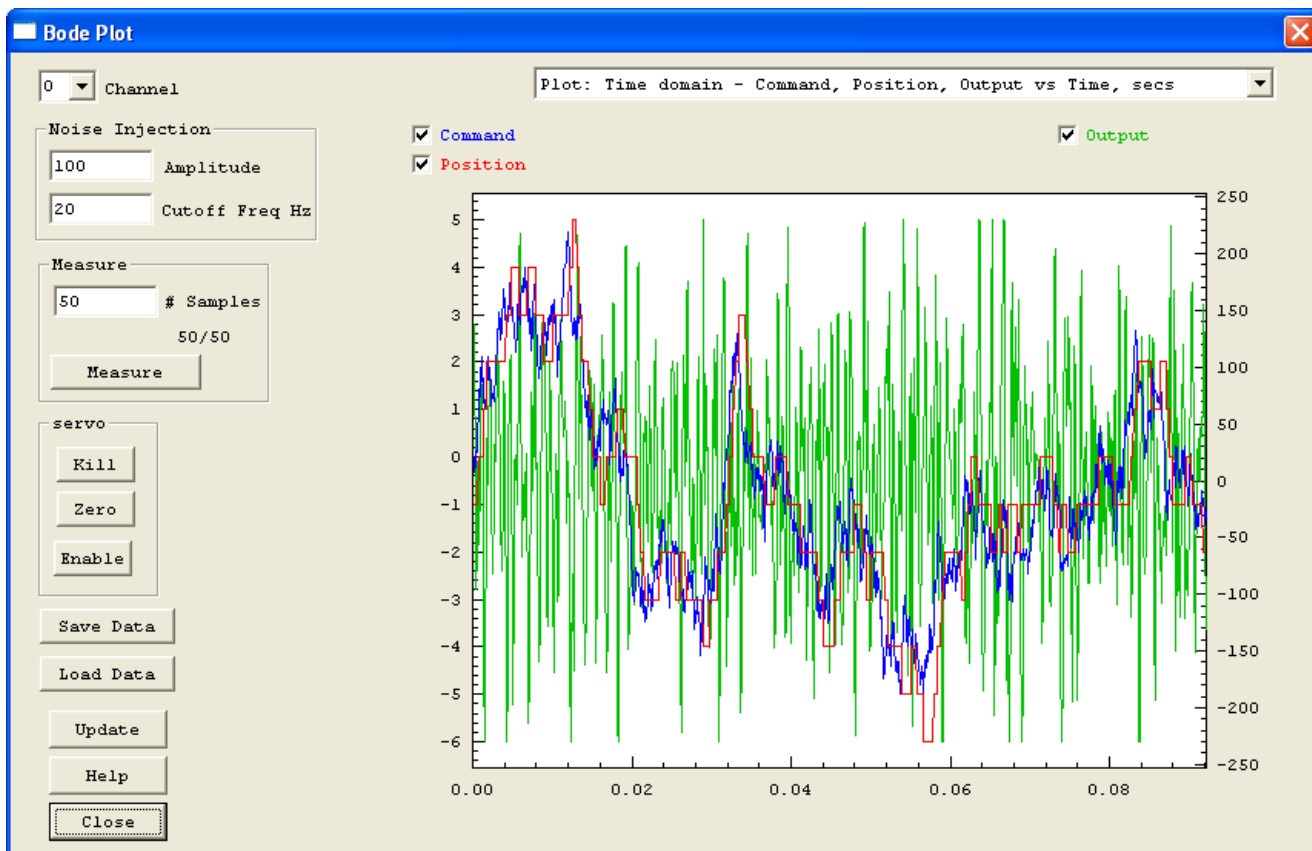
Plot: Open Loop - Magnitude and Phase vs Freq, Hz
Plot: Open Loop - Magnitude and Phase vs Freq, Hz
Plot: Time domain - Command, Position, Output vs Time, secs

A typical time domain measurement is shown below. The *blue* graph shows the random stimulus to the system. The *red* graph shows the system's response, which in this example is the output of an encoder. Note that the position is quantized to integer counts and has a range of approximately 10 counts. This is nearly the minimum number of counts to expect a reasonable Bode Plot measurement. A larger range of encoder counts could be obtained by driving the system harder by increasing the Noise Injection Amplitude, provided there is additional output drive available.

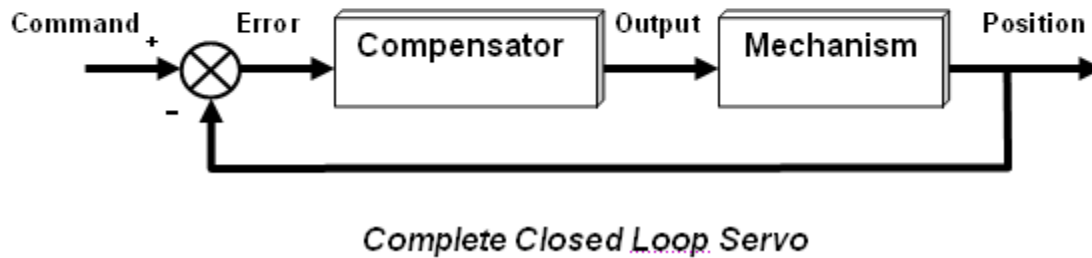
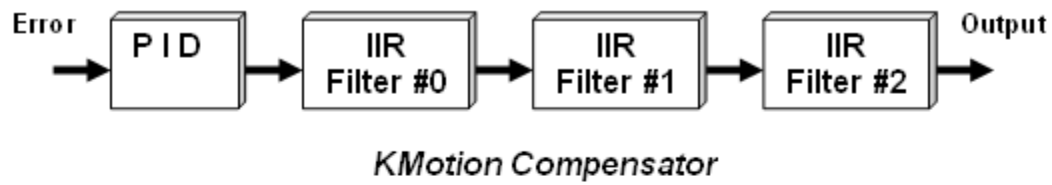


The graphs below include the output drive signal shown in *green*. Note that the output drive is on the verge of saturating at the maximum allowed output value (example is from a 3 phase brushless motor, maximum allowed PWM counts of 230). This shows that we are driving the system as hard as possible without saturating in order to obtain the most accurate measurement.

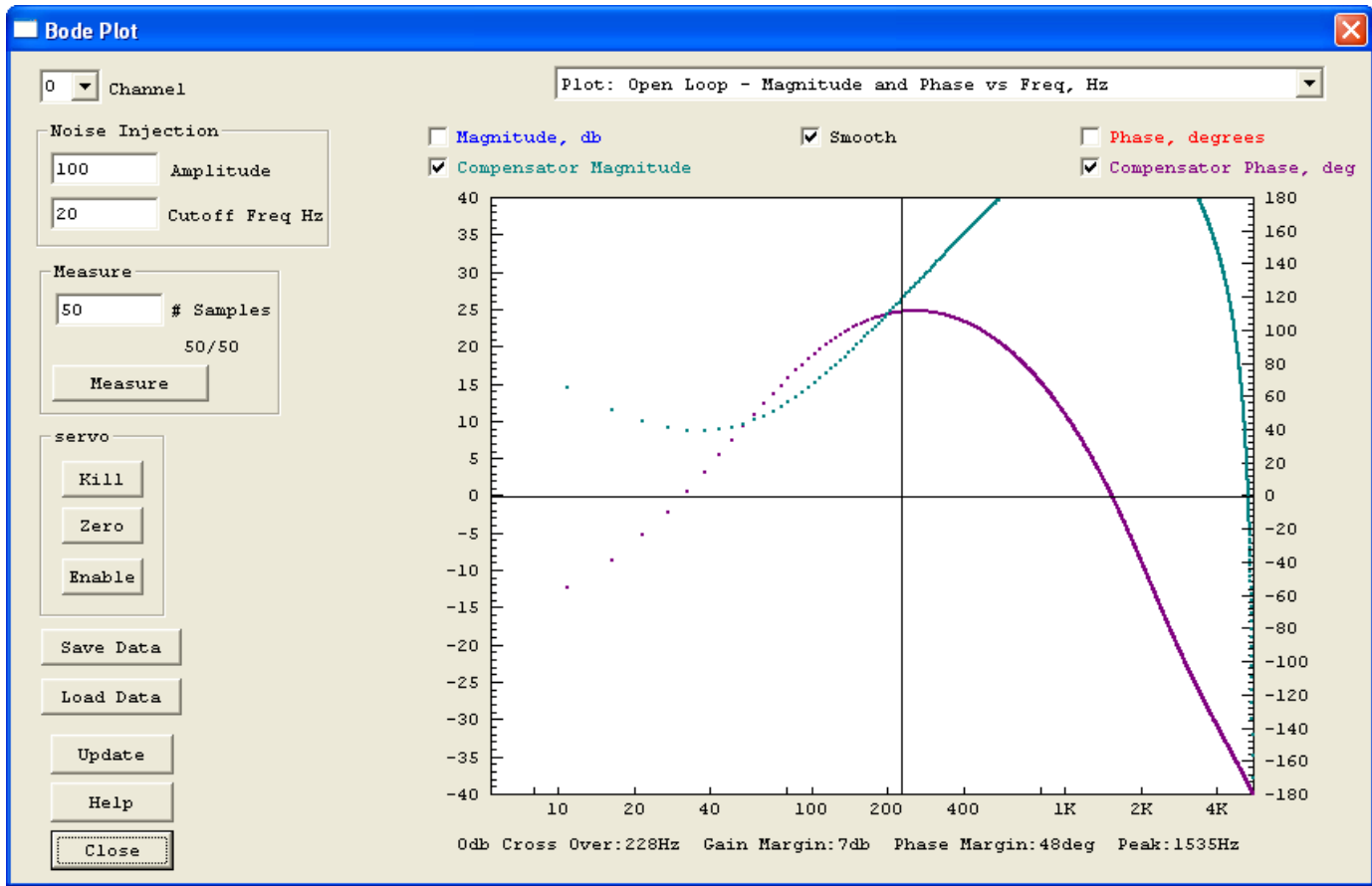
Another means of improving the measurement accuracy (as well as servo performance in general) is obviously to increase the encoder resolution if economically or otherwise feasible.



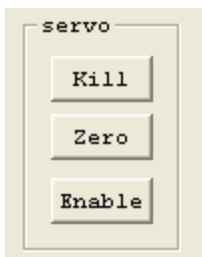
Compensator Response



The *Bode Plot Screen* is also capable of graphing a *Bode Plot* of the combined PID and IIR Filters. This is often referred to as the *Compensator* for the system. The Cyan graph shows the magnitude of the compensator and the violet graph shows the phase of the compensator. Notice that in this example the maximum phase has been adjusted to match the 0 db crossover frequency of the system (~ 233 Hz) to maximize the system phase margin.

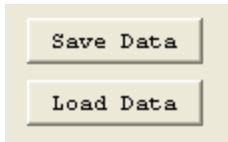


Axis Control



The Axis Control buttons are present to conveniently disable (Kill), Zero, or Enable an axis. If the axis becomes unstable (possible due to a gain setting too high), the Kill button may be used to disable the axis, the gain might then be reduced, and then the axis may be enabled. The Enable button downloads all parameters from all screens before enabling the axis in the same manner as the [Measure](#) button described above.

Note for brushless output modes that commutate the motor based on the current position, Zeroing the position may adversely affect the commutation.



Save/Load Data

The Save/Load Data buttons allow the captured Bode Plot to be saved to a text file and re-loaded at a later time. The text file format also allows the data to be imported into some other program for display or analysis. The file format consists of one line of header followed by one line of 9 comma separated values, one line for each frequency. The values are:

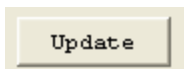
1. Frequency in Hz
2. Input Stimulus - Real Part of complex number
3. Input Stimulus - Complex Part of complex number (always zero)
4. Measured Closed Loop Output - Real Part of complex number
5. Measured Closed Loop Output - Complex Part of complex number
6. Open loop Magnitude - in decibels
7. Open loop Phase - in degrees
8. Open loop Magnitude - in decibels - "smoothed"
9. Open loop Phase - in degrees - "smoothed"

Example of data file follows:

```
Freq,InputRe,InputIm,OutputRe,OutputIm,Mag,Phase,SmoothMag,SmoothPhase
0,2.329706e+007,0,2.344316e+007,0,44.10739,-180,0,0
5.425347,1.968735e+007,0,1.98995e+007,-32055.19,39.34598,-171.5001,39.34598,-171.5001
10.85069,1.816919e+007,0,1.848909e+007,-713239.6,27.48402,-116.3662,29.58283,-139.1553
16.27604,2.024904e+007,0,2.124962e+007,-1383543,21.91849,-129.5997,25.14718,-134.5283
21.70139,1.53403e+007,0,1.645491e+007,-1651059,18.38331,-129.7526,22.70204,-127.6139
27.12674,1.225619e+007,0,1.301412e+007,-1336369,18.60411,-125.4229,20.60267,-123.6751
32.55208,7014539,0,7393482,-958714.3,17.18516,-118.9553,18.9778,-119.2762
```

▪
▪
▪

Update Pushbutton



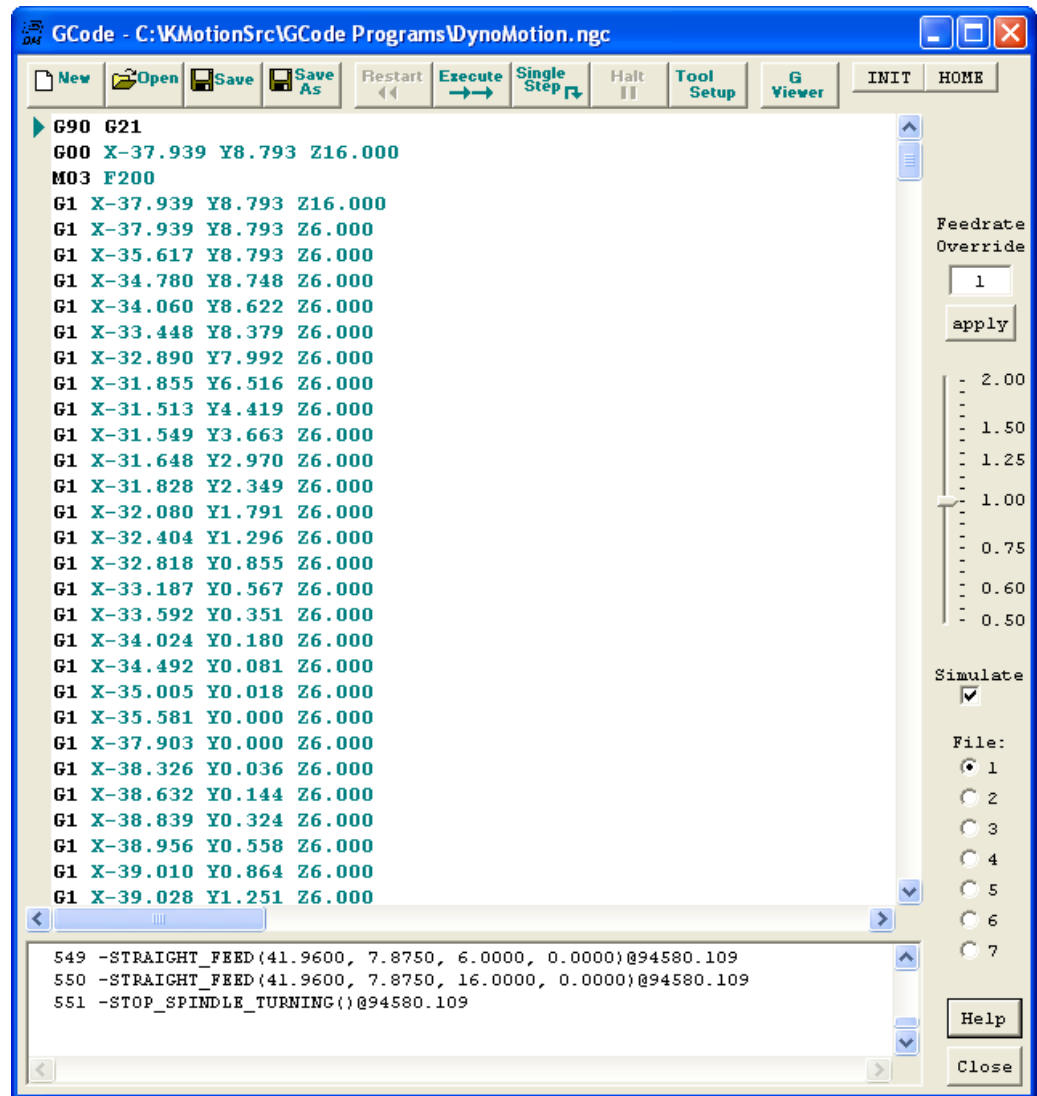
The Update button may be used to update the displayed [compensator](#) graph if any parameters (on other screens) have been modified and not downloaded or otherwise acted upon. If Measure, Download, Step, Move, Save, Close, or Enable is used on this or the any other screen then this command is unnecessary, however if none of these commands is performed, then this button may be used to update the graph.

G Code Quick Reference

G Codes

G0 X3.5 Y5.0 Z1.0
 A2.0 (Rapid move)
G1 X3.5 Y5.0 Z1.0
 A2.0(linear move)
G2 X0.0 Y0.5 I0 J0.25
 (CW Arc move)
G3 X0.0 Y0.5 I0 J0.25
 (CCW Arc move)
G4 P0.25
 (Dwell seconds)
G10L2Pn
 G10L2P1X0Y0Z0
 (Set Fixture Offset #n)
G20 Inch units
G21 mm units
G28 Move to
 Reference Position #1
G30 Move to
 Reference Position #2
G40 Tool Comp Off
G41 Tool Comp On
 Left of Contour)
G42 Tool Comp On
 (Right of Contour)
G43 Hn (Tool #n length
 comp On)
G49 (Tool length comp
 off)
G53 Absolute Coord
G54 Fixture Offset 1
G55 Fixture Offset 2
G56 Fixture Offset 3
G57 Fixture Offset 4
G58 Fixture Offset 5
G59 Fixture Offset 6
G59.1 Fixture Offset 7
G59.2 Fixture Offset 8
G59.3 Fixture Offset 9
G90 Absolute
 Coordinates
G91 Relative
 Coordinates
G92 Set Global Offset

G Code Screen



[Show screen feature descriptions](#)

See Also [G Code Viewer Screen](#) and [Tool Setup Screen](#)

The **G Code Screen** allows the user to edit G Code Programs and execute them.

GCode is a historical language for defining Linear/Circular/Helical Interpolated Motions often used to program numerically controlled machines (CNC Machines).

See the Quick Reference to the left for commonly used G Code commands.

Coordinates G92 X0Y0
Z0

M Codes:

M0 (Program Stop)

M2 (Program End)

M3 Spindle CW

M4 Spindle CCW

M5 Spindle Stop

M6 Tool Change

M7 Mist On

M8 Flood On

M9 Mist/Flood Off

M98 Pxxx Call

Subroutine

M99 Return from

Subroutine

Other Codes:

F (Set Feed rate in/min
or mm/min)

S (Spindle Speed)

D (Tool)

O Subroutine Label

Comments:

(Simple Comment)

(MSG,OK toContinue?)

(CMD,EnableAxis0)

(BUF,SetBitBuf29)

KMotion's G Code interpreter was derived from the Open Source EMC G Code Interpreter. Click here for the [EMC User Manual](#) (Only the G Code portions of the manual, Chapters 10-14 pertain to **KMotion** G Code)

Specially coded comments embedded within a GCode program may be used to issue **KMotion** Console Script commands directly to **KMotion**.

A comment in the form: (CMD,xxxxxx) will issue the command xxxxxx immediately to **KMotion** as soon as it is encountered by the Interpreter. Any **KMotion** command that does not generate a response may be used in this manner.

A comment in the form: (BUF,xxxxxx) will place the command xxxxxx into **KMotion's** coordinated motion buffer. Coordinated motion sequences are download to a motion buffer within **KMotion** before they are executed. This guarantees smooth uninterrupted motion. The BUF command form allows a command to be inserted within the buffer so they are executed at an exact time within the motion sequence. Only the following **KMotion** Script commands may be used in this manner.

SetBitBuf, ClearBitBuf, SetStateBitBuf.

Additionally, a comment in the form: (MSG,xxxxxx) will pause GCode Execution and display a pop-up message window displaying the message xxxxxxxx.

C Program Screen

Constants:

FALSE 0

TRUE 1

PI

3.14159265358979323846264

PI_F 3.1415926535f

TWO_PI (2.0 * PI)

TWO_PI_F (2.0f * PI_F)

PI_2F (PI_F * 0.5f)

TRAJECTORY_OFF 0

TRAJECTORY_INDEPENDENT

1

TRAJECTORY_LINEAR 2

TRAJECTORY_CIRCULAR 3

TRAJECTORY_SPECIAL 4

Axis Input Modes

ENCODER_MODE 1

ADC_MODE 2

RESOLVER 3

USER_INPUT_MODE 4

Axis Output Modes

MICROSTEP_MODE 1

DC_SERVO_MODE 2

BRUSHLESS_3PH_MODE 3

BRUSHLESS_4PH_MODE 4

DAC_SERVO_MODE 4

Data Gather/Plot Functions:

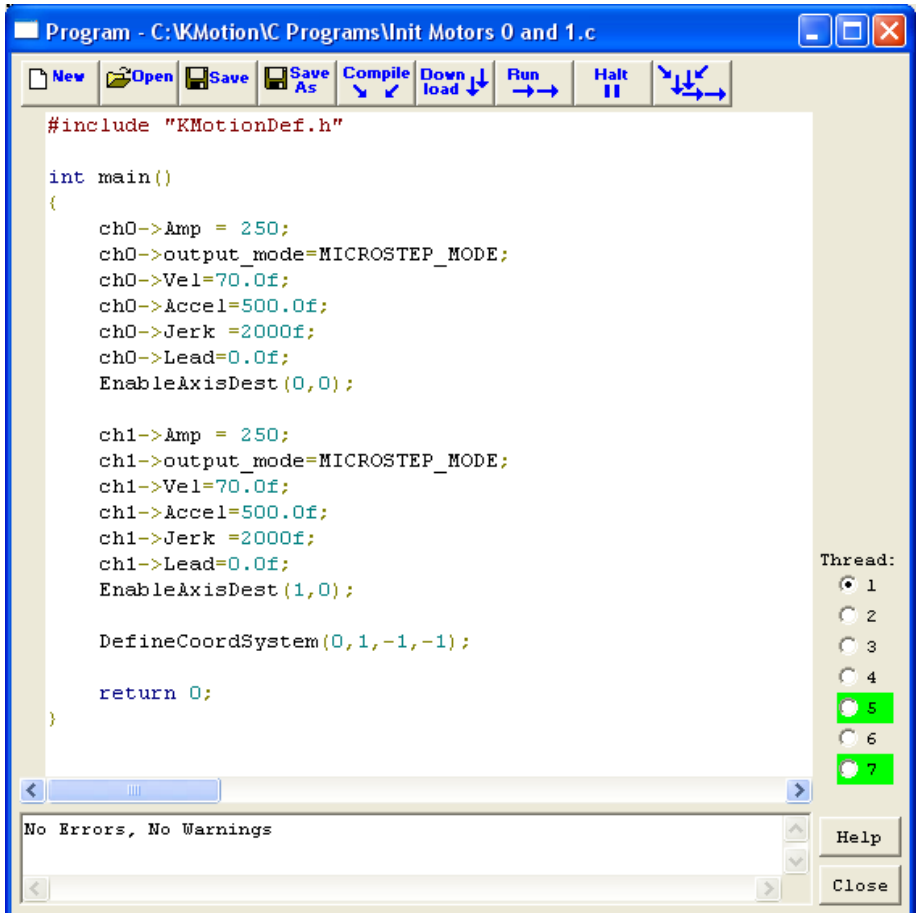
void

**SetupGatherAllOnAxis(int c,
int n_Samples);**

void TriggerGather();

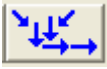
int CheckDoneGather();

Analog I/O Functions:



The **C Program Screen** allows the user to edit C language programs, compile, link, download, and run them within the **KMotion** board. C programs executing within the **KMotion** board have direct access to all the Motion, I/O, and other miscellaneous functions incorporated into **KMotion** System.

One of the most powerful features of the **KMotion** system is the ability for a user to compile and download native DSP programs and have them run in real time. Native DSP code runs faster than interpreted code. The TMS320C67x DSP that powers the **KMotion** system has hardware support for both 32 bit and 64 bit floating point math. Multiple threads (programs) may execute simultaneously (up to 7). The integrated C compiler allows with

a single pushbutton  to save, compile, link, download, and execute all within a fraction of a second. After programs have been developed and tested they may be [flashed into memory](#) and run stand alone with no host connection.

ADC(ch);
DAC(ch, value);

Power Amp Functions:

void WritePWMR(int ch, int v);
void WritePWM(int ch, int v);
void Write3PH(int ch, float v,
double angle_in_cycles);
void Write4PH(int ch, float v,
double angle_in_cycles);

Timer Functions:

double Time_sec();
void WaitUntil(double
time_sec);
void Delay_sec(double sec);
double
WaitNextTimeSlice(void);

Axis Move Functions:

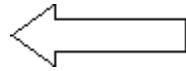
void DisableAxis(int ch);
void EnableAxisDest(int ch,
double Dest);
void EnableAxis(int ch);
void Zero(int ch);
void Move(int ch, double x);
void MoveRel(int ch, double
dx);
int CheckDone(int ch);
void MoveXYZ(double x,
double y, double z);
int CheckDoneXYZ();
void DefineCoordSystem(int
axisx, int axisy,
int axisz, int axis a);

Digital I/O Functions:

void SetBitDirection(int bit,

Other features of the **C Program Screen** include a rich text editor with syntax highlighting, keyword [drop down lists](#), [function tips](#), unlimited [undo/redo](#), and [Find/Replace with regular expressions](#).

See list on left for available constants and functions.



For a more details on the functions, see the [KMotionDef.h](#) header file. This file is normally included into a user program so that all accessible base functions and data structures are defined.

See [PC-DSP.h](#) for common definitions between the PC host and *KMotion* DSP.

```
int dir);
int GetBitDirection(int bit);
void SetBit(int bit);
void ClearBit(int bit);
void SetStateBit(int bit, int
state);
int ReadBit(int bit);
```

Print to Console Screen

Functions:

```
int printf(const char *format,
...);
```

Print to Windows File

Functions:

```
FILE *fopen(const char*,
const char*);
int fprintf(FILE *f, const char
* format, ...);
int fclose(FILE *f);
```

Thread Functions:

```
void StartThread(int thread);
void PauseThread(int thread);
void ThreadDone();
int ResumeThread(int
thread);
```

Math Functions:

```
double sqrt(double x);
double exp(double x);
double log(double x);
double log10(double x);
double pow(double x, double
y);
double sin(double x);
double cos(double x);
double tan(double x);
double asin(double x);
double acos(double x);
double atan(double x);
```


**double atan2(double y,
double x);**

float sqrtf (float x);

float expf (float x);

float logf (float x);

float log10f(float x);

float powf (float x, float y);

float sinf (float x);

float cosf (float x);

float tanf (float x);

float asinf (float x);

float acosf (float x);

float atanf (float x);

float atan2f(float y, float x);

IIR Filter Screen

The screenshot shows the 'Filters' window with three filter configuration panels. Filter 0 is set to 'Pole-Zero' and shows the transfer function $\frac{y(s)}{x(s)} = \frac{(1+s/\omega_{n1})(1+s/\omega_{n2})}{(1+s/\omega_{d1})(1+s/\omega_{d2})}$. Filter 1 is set to 'Z Domain' and shows the transfer function $\frac{y(z)}{x(z)} = \frac{B_0 + B_1z^{-1} + B_2z^{-2}}{1 - A_1z^{-1} - A_2z^{-2}}$. Filter 2 is set to 'Low Pass' and shows the transfer function $\frac{y(s)}{x(s)} = \frac{1}{1+s/\omega}$. Each panel has a 'Compute' button and a 'Clear' button. At the bottom, there is a 'Channel' dropdown, a 'C Code -> Clipboard' button, a 'Download' button, a 'Help' button, and a 'Close' button.

The **IIR Filter Screen** allows setting various IIR (Infinite Impulse Response) *Filters* into the control loop. **KMotion** allows up to three - 2nd order bi-quadratic stages per axis to be connected in cascade. See the [KMotion Servo Flow Diagram](#) for the placement of the IIR Filters.

KMotion implements the filters using Z-domain coefficients shown on the bottom half of the screen. Because of the common confusion of the names and signs of the coefficients, the transfer function form is shown for reference.

$$\frac{y(z)}{x(z)} = \frac{B_0 + B_1z^{-1} + B_2z^{-2}}{1 - A_1z^{-1} - A_2z^{-2}}$$

Note that setting B0=1.0 and all other coefficients, B1, B2, A1, and A2, to zero causes a transfer function of unity, effectively bypassing the filter. A Clear pushbutton is available to conveniently set this mode.

B0	B1	B2
1	0	0
Clear	0	0
	A1	A2

The top portion of each filter section allows various common filters to be specified in the s-domain. Supported filter types are: [1st order Low Pass](#), [2nd Order Low Pass](#), [Notch](#), and two real [poles and zeros](#) and are selected using a drop-down list box.. Z-domain is a place holder used as a reminder that the z-domain coefficients were determined directly by some other means. The form of each of the filters in the s-domain is shown below.

Low Pass

Low Pass
 Low Pass 2nd
 Notch
 Pole-Zero
 Z Domain

Low Pass

$$\frac{y(s)}{x(s)} = \frac{1}{1 + s/\omega}$$

Low Pass 2nd

$$\frac{y(s)}{x(s)} = \frac{1}{s^2/\omega_n^2 + Qs/\omega_n + 1}$$

Notch

$$\frac{y(s)}{x(s)} = \frac{s^2 + 2s\omega_n + \omega_n^2}{s^2 + 2\eta s\omega_n + \omega_n^2}$$

Pole-Zero

$$\frac{y(s)}{x(s)} = \frac{(1 + s/\omega_{z1})(1 + s/\omega_{z2})}{(1 + s/\omega_{p1})(1 + s/\omega_{p2})}$$

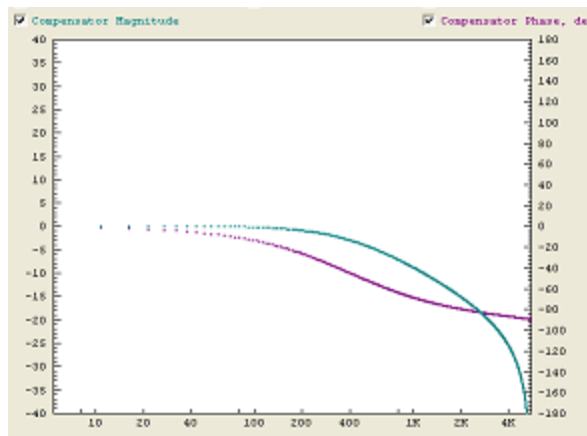
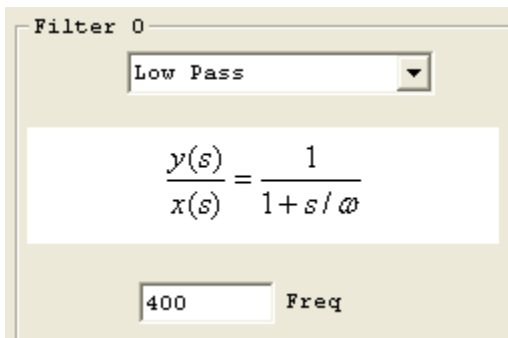
If an s-domain filter type is selected and its corresponding parameters specified, then depressing the *Compute* pushbutton will convert the s-domain transfer function to the z-domain using Tustin's approximation (with frequency pre-warping) and automatically fills in the z-domain coefficients. Note that **KMotion** always utilizes the current (most recently computed or entered) z-domain coefficients, regardless of any changes that might be made to the s-domain section.

Note that the [Bode Plot Screen](#) has the capability to graph the combined transfer function of all three IIR filters and the PID filter. This is referred to as the *Servo Compensation*. To view the transfer function of a *single* IIR filter, set the other filter and PID sections to unity (for PID set P=1, I=0, D=0 or for an IIR Filter B0=1, B1= B2=A1=A2=0).

Below are examples of each of the s-domain filter types (shown individually):

Low Pass (1st order)

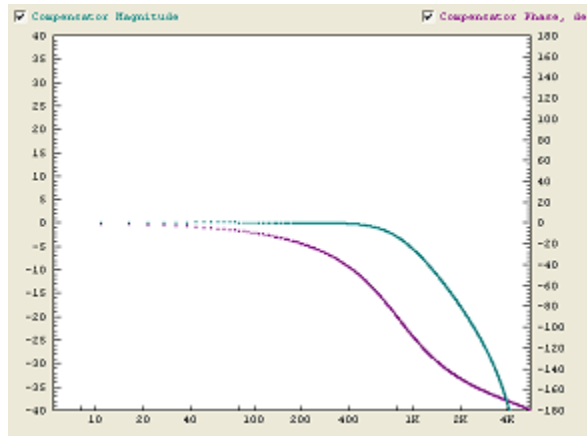
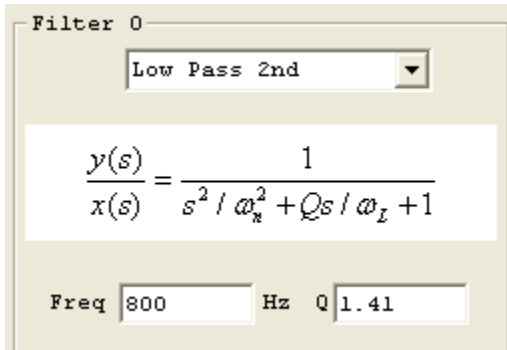
A *Low Pass* filter is commonly used in a servo system to reduce high frequency noise (or spikes) in the output. It also has the desirable effect of decreasing the gain at high frequencies. If a system's gain at high frequencies is increased sufficiently to reach 0 db it may become unstable. Unfortunately it has the effect of introducing phase lag (negative phase) which will reduce the [phase margin](#). A 1st order Low Pass filter has 45 degrees phase and attenuation of -3db at the specified cutoff frequency. The cutoff frequency should normally be specified much higher than the servo bandwidth in order to have only a small phase lag at the system's bandwidth.



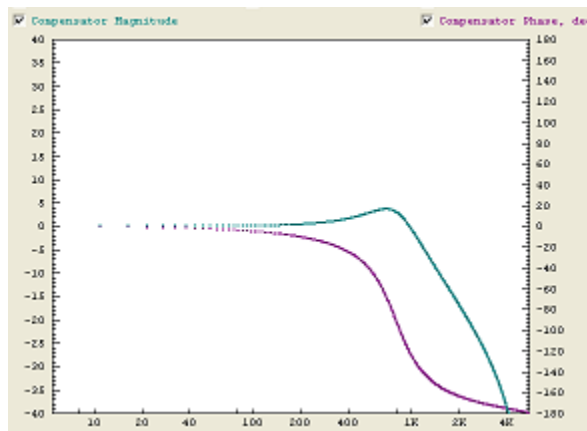
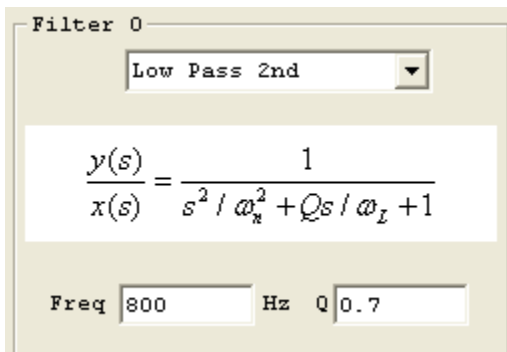
Low Pass (2nd order)

A *2nd order Low Pass* filter is commonly used in a similar manner as a [1st order low pass filter](#), except that it has higher attenuation than a 1st order low pass filter. Unfortunately it also introduces more phase lag than a 1st order low pass filter. In most cases the cutoff frequency for a 2nd order low pass filter will have to be specified at a higher frequency than a 1st order filter, in order to have similar phase lag at the system bandwidth frequency. Even so, the 2nd order low pass filter is usually preferable in that it provides slightly more high frequency attenuation and "smoothing" of the output.

A 2nd order Low Pass filter also allows a Q parameter which changes the sharpness of the filter. A smaller value of Q results in a sharper filter at the expense of some peaking (gain increases before decreasing). A Q value of 1.41 (called a Butterworth filter), shown immediately below, is the minimal value that may be specified without peaking.



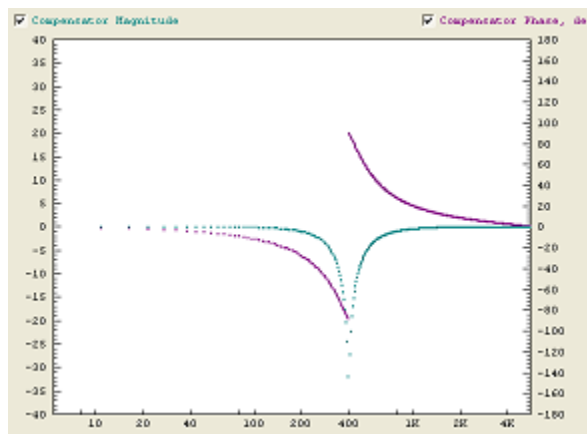
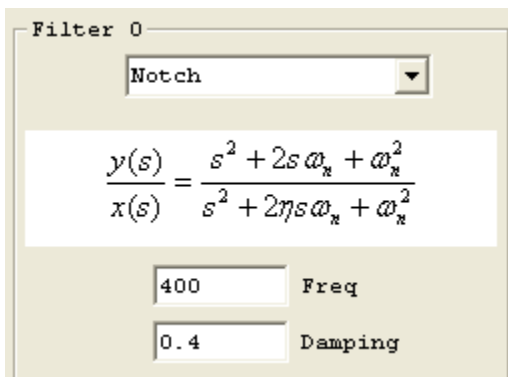
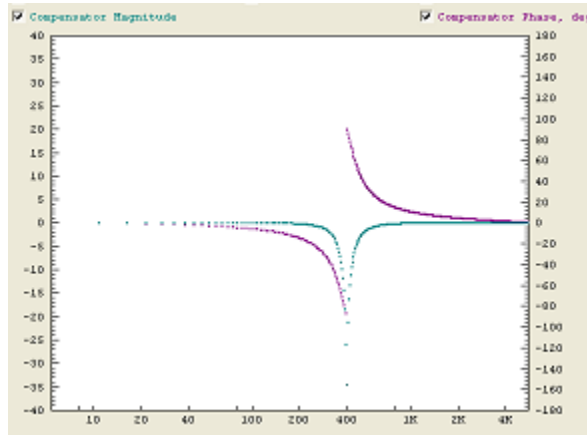
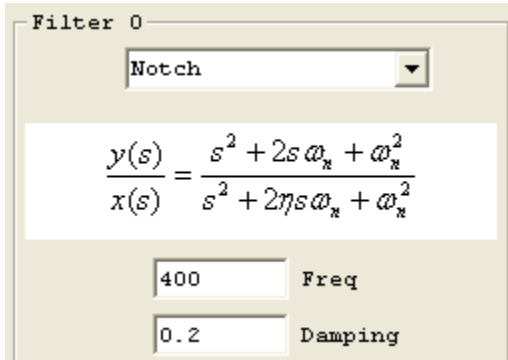
A Q value of 0.7 shows "peaking".



Notch

A *Notch* filter is commonly used in servo system when a sharp mechanical resonance would otherwise cause a system to become unstable. A Notch filter is able to attenuate a very localized range of frequencies. It has a damping factor, η , which effects sharpness or width of the notch. The disadvantage of using a notch filter is some phase lag which tends to decrease [phase margin](#). The introduced phase lag will be less the narrower the notch is (less damping), as well as the distance the notch frequency is above the system bandwidth.

Shown below are two notch filters both at 400 Hz, one with 0.2 damping and the other with 0.4 damping.



Pole-Zero

A *Pole-Zero* filter is commonly used in a lead-lag configuration shown below to shift the phase positive at the 0 db crossover frequency of the system in order to increase [phase margin](#). The filter shown has maximum positive phase of approximately 80 degrees at 200 Hz. This is accomplished by setting the N1,N2 (zeros or numerator frequencies) 2X *lower* than 200 Hz (100 Hz), and the D1,D2 (poles or denominator frequencies) 2X *higher* than this (400 Hz). A lead-lag filter (or compensator) may often be used in place of derivative gain (in the PID stage), and has a significant advantage of lower high frequency gain. If a system's gain at high frequencies is increased sufficiently to reach 0 db it may become unstable.

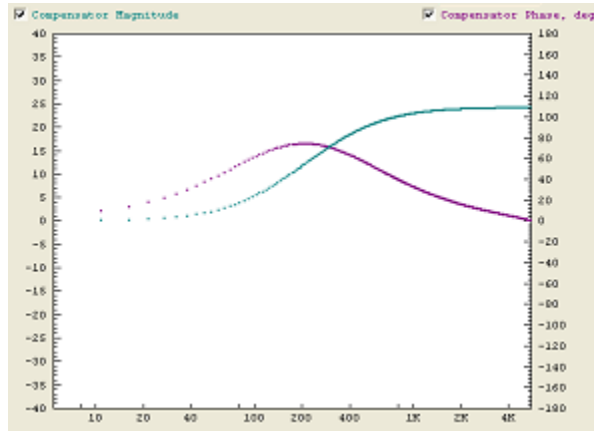
Filter 0

Pole-Zero

$$\frac{y(s)}{x(s)} = \frac{(1+s/\omega_{n1})(1+s/\omega_{n2})}{(1+s/\omega_{d1})(1+s/\omega_{d2})}$$

N1 100 N2 100 Hz

D1 400 D2 400 Hz



Copy C Code to Clipboard

C Code ->
Clipboard

This pushbutton causes the current z-domain filter coefficients to be copied to the clipboard in a form that may be pasted directly into a **KMotion** C Program, see also the [C Program Screen](#).

```
ch0->iir[0].B0=232.850006;
ch0->iir[0].B1=-450.471008;
ch0->iir[0].B2=217.869995;
ch0->iir[0].A1=1.001990;
ch0->iir[0].A2=-0.250994;
```

```
ch0->iir[1].B0=1.000000;
ch0->iir[1].B1=0.000000;
ch0->iir[1].B2=0.000000;
ch0->iir[1].A1=0.000000;
ch0->iir[1].A2=0.000000;
```

```
ch0->iir[2].B0=0.175291;
ch0->iir[2].B1=0.350583;
ch0->iir[2].B2=0.175291;
ch0->iir[2].A1=0.519908;
ch0->iir[2].A2=-0.221073;
```

Download

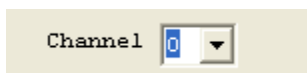
Download

The *Download* push button downloads the filters for the selected axis (along with all axis configuration and tuning parameters) to the **KMotion**.

Configuration and FLASH Screen

(Click on above image to jump to relative topic)

The **Configuration and FLASH Screen** displays and allows changes to **KMotion's** configuration and allows the configuration, new firmware, or user programs to be FLASH'ed to non volatile memory.



Each axis channel is configured independently. To view or make changes to a configuration first select the desired axis channel using the channel drop down. Note that changing an axis on any screen switches the active channel on all other screens simultaneously.

The parameters for each axis's configuration are grouped into three classes: Definitions, Tuning, and Filters. Each class of parameters are displayed on three corresponding screens:

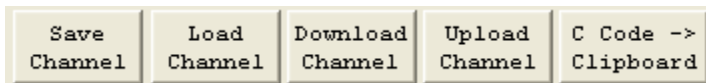
- Configuration Screen
- Step Response Screen
- IIR Filter Screen

The *Configuration Screen* contains *definition* parameters that should be set once and remain set unless a physical change to the hardware is made. For example, a Stepper motor might be replaced with a Brushless Motor and Encoder.

The *Step Response Screen* contains parameters that are *tuning* related and are located where the tuning response is most often adjusted and checked. For example, PID (proportional, integral) parameters are located there.

The *IIR Filter Screen* contains parameters related to servo *filters*.

Utilities

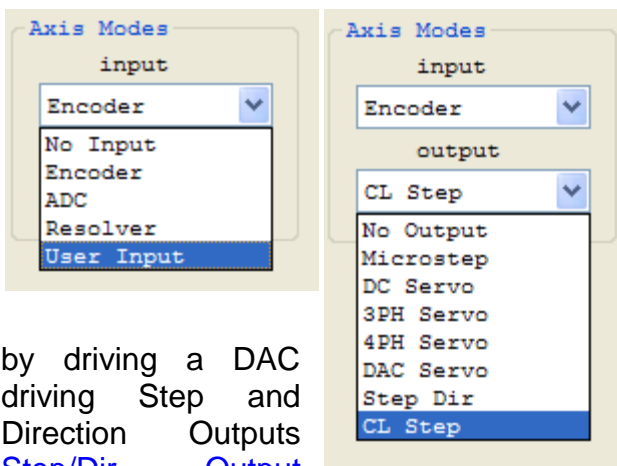


The buttons along the bottom of the Configuration Screen allow a set of axis parameters to be:

- Saved or Loaded from a disk file (*.mot)
- Uploaded or Downloaded to a **KMotion**
- Converted to equivalent C Code for use in a [KMotion C Program](#)

Note that these buttons operate on all parameters (for one axis) from all screens as a unit.

Axis Modes



by driving a DAC
driving Step and
Direction Outputs
[Step/Dir](#) [Output](#)

Use the respective dropdown to set either the axis Input or Output Mode. The input mode defines the type of position measurement (if required) for the axis. Closed loop control always requires some type of position measurement. For open loop stepper motor control, position measurement is optional. The output mode determines how the output command should be achieved. Either by driving the on board PWMs and Full Bridge Drivers to control a specific type of motor, signal that will drive an external power amplifier, or by Direction digital outputs. For External Step and see [Step and Direction Output Mode](#) and [Closed Loop Mode](#).

Input Channels

		gain	offset
0	0	1	0
1	1	1	0

The Input Channels section specifies which channels for the selected input device will be used. Some Input Modes require two devices to be specified and some Input Modes only require one device. If the selected Input Mode only requires one device then the second Input Channel (Input Chan 1) is not used and may be set to any number. This may be the channel of an Encoder input or an ADC input depending on the

selected input mode. Resolvers requires two ADC input channels (for sine and cosine), for all other modes the second channel number is not used.

The gain and offset parameters are applied to the respective input device. The gain is applied before the offset, i.e. $x' = ax + b$, where a is the gain and b is the offset..

Incremental encoders only utilize the gain parameter which may be used to scale or reverse (using a negative gain) the measurement.

A *Resolver* is a device that generates analog sine and cosine signals as its shaft angle changes. Either one or multiple sine and cosine waves may be produced per revolution of a resolver. An encoder that generates analog sine and cosine signals may also be connected to a **KMotion** as though it was a resolver. Resolver inputs may utilize both gains and offsets and be adjusted such that the sine and cosine ADC measurements are symmetrical about zero and have the same amplitude. Gain and offset errors may be introduced by either the ADC input circuitry and/or the Resolver itself. If one were to plot the sine vs. cosine signals as a resolver's position changes, the result should be circle. **KMotion** computes the arctangent of the point on the circle (also keeping track of the number of rotations) to obtain the current position. An offset or elliptical "circle" will result in a distorted position measurement throughout the cycle. Therefore note that adjusting the gains and offsets will result in changing the linearity of the position measurement, not the scale of the position measurement itself. The scale of a resolver input will always be 2π radians per cycle.

An ADC input uses a single absolute ADC channel input to obtain the position measurement. Gain0 and Offset0 may be used to modify the ADC counts from -2048 .. +2047 to a desired range.

Output Channels

		gain	offset
0	0	1	0
1	1		

The Output Channels section specifies which channels for the selected output device will be used. Some Output Modes require two devices to be specified and some Output Modes only require one device. For Output modes that only require one output device the second device will be disabled. If the selected Output Mode only requires one device then the second Output Channel (Output Chan 1) is not used and may

be set to any number. The specified output device may be the channel of a [PWM](#) connected to an on-board power amplifier, a Step/Direction Generator, or a DAC that is used to drive an external power amplifier.

Stepper mode and 4 phase brushless mode require two channels of PWM to be specified.

DC Servo motor (Brush motor type) only require one PWM channel.

3 Phase brushless motors require a consecutive pair of PWM channels. In 3 Phase output mode, only the Output Channel 0 value is used and must be set to an even PWM number.

For Step and Direction output mode and CL Step (Closed Loop Step/Dir), the output channel 0 is used to specify which Step/Direction Generator will be used and drive mode (active high/low or open collector) will be used. Each Step/Direction Generator has assigned I/O Pins. See [Step and Direction Output Mode](#).

Some output devices support the application of a gain and offset. See also the related Console Commands [OutputGain](#) and [OutputOffset](#).

Microstepper Amplitude, Max Following Error, Inv Dist Per Cycle, Lead Compensation

Microstepper Amplitude	100
Max Following Error	10000000
Inv Dist Per Cycle	1
Lead Compensation	0

Microstepper Amplitude is only applicable to configurations with output mode of Microstepper. This parameter sets the amplitude (of the sine wave) in PWM counts (0 .. 255) that will be output to the sine and cosine PWM channels while moving slowly or at rest. Note that at higher speeds **KMotion** has the ability to increase the amplitude to compensate for motor inductance effects and *may* actually be higher. See *Lead Compensation* in this same section.

Max Following Error is applicable to all closed loop servo output modes (DC Servo, 3 Phase Brushless, 4 Phase brushless, and DAC Servo). Whenever the commanded destination and the measured position differ by greater than this value, the axis will be disabled (if this axis is a member of the defined coordinate system, then any coordinated motion will also stop). To disable following errors set this parameter to a large value.

Inv Dist Per Cycle applies to Stepper, 3 Phase, and 4 Phase motors. For a stepper motor, the *distance per cycle* defines the distance that the commanded destination should change by for a motor coil to be driven through a complete sinusoidal cycle. Parameter should be entered as the inverse (reciprocal) of the distance per cycle. Stepper motors are most often characterized by shaft angle change per "Full Step". A motor coil is driven through a complete cycle every four - "Full Steps". See the following examples:

Example #1 : A mechanism moves 0.001" for each full step of a step motor It is desired for commanded distance to be in inches.

Result: One Cycle = 4 full steps = 0.004", Thus $\text{InvDistPerCycle} = 1.0/0.004 = 250.0$ (cycles/inch). Commanding a move of 1.00 will generate 250 sine waves, or the equivalent of 1000 full steps, or one inch of movement.

Example #2 : *InvDistPerCycle* is left at the default value of 1.0

Result: Move units are in cycles. Commanding a move of 50 will generate 50 sine waves, or the equivalent of 200 full steps, or one revolution of a 200 Step or 1.8 degree motor.

For 3 Phase or 4 Phase motors, *Inv Dist Per Cycle* represents the inverse of the distance for one complete commutation cycle. See the example below.

Example #3 : A 3 phase motor/encoder has a 4096 count per revolution encoder which is used for position feedback and for motor commutation. InputGain0 is set to 1.0 so position measurement remains as encoder counts. The motor design is such that the commutation goes through 3 complete cycles each motor revolution.

Result: One Cycle = 4096 counts/3.0 Thus InvDistPerCycle = 3.0/4096 = 0.000732421875.

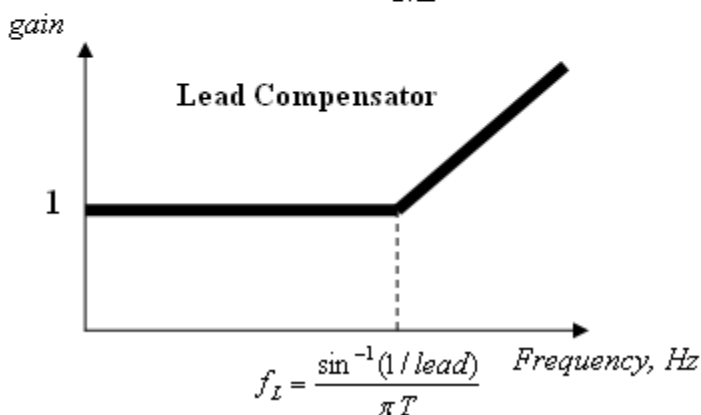
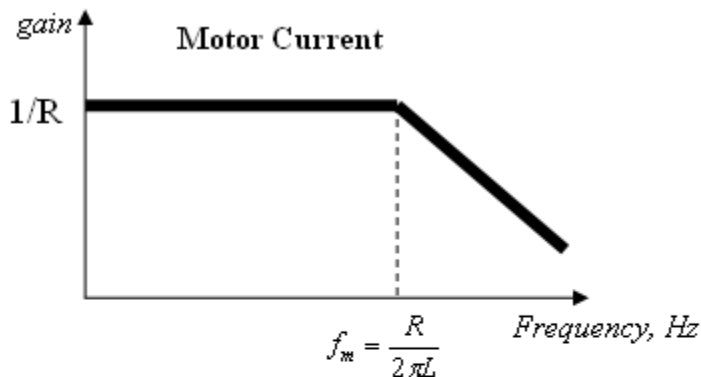
Note that it is important to use a high degree of precision to avoid commutation errors after moving to very large positions (or at constant velocity for a long period of time). **KMotion** maintains *Inv Dist Per Cycle* (as well as position) as a double precision (64 bit) floating point number for this reason (*more than 70 years* at 1 MHz would be required to have 1 count of error)

Lead Compensation may be used to compensate for motor inductance. When a voltage is applied to a coil at a low frequencies, the current flow is dictated by the coil's resistance and is constant. As

$$f_m = \frac{R}{2\pi L}$$

the frequency increases at some point, where , the inductance, L , begins to dominate and the current drops (see plot below). **KMotion's** Lead Compensator has the opposite effect, it has a constant gain of 1 and at some point increases with frequency. The Lead Compensation parameter sets (indirectly) the frequency where this occurs. If the frequency is set to match the frequency of the motor, the effects will cancel, and the motor current (and torque) will remain constant to a much higher frequency.

This assumes that the nominal drive voltage is lower than the available supply voltage. For example, a 5V stepper motor might be driven with a 15V supply to allow head room for the applied voltage to be increased at high frequencies (speeds).



The simple formula that implements the Lead Compensation is:

$$v' = v + \Delta v L$$

where v is the voltage before the compensation, v' is the voltage after the compensation, Δv is the change in output voltage from the last servo sample, and L is the Lead Compensation value.

The following formula will compute the "knee" frequency for a particular *lead* and servo sample rate (normally $T=90$ us).

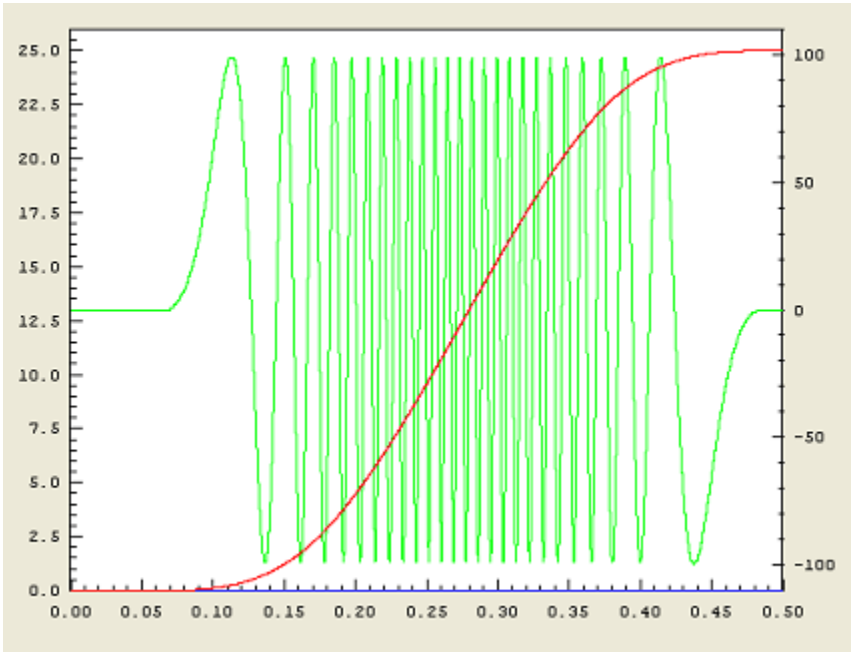
$$f_L = \frac{\sin^{-1}(1/\text{lead})}{\pi T}$$

or the inverse of this formula will provide the lead value to position the knee at a particular frequency.

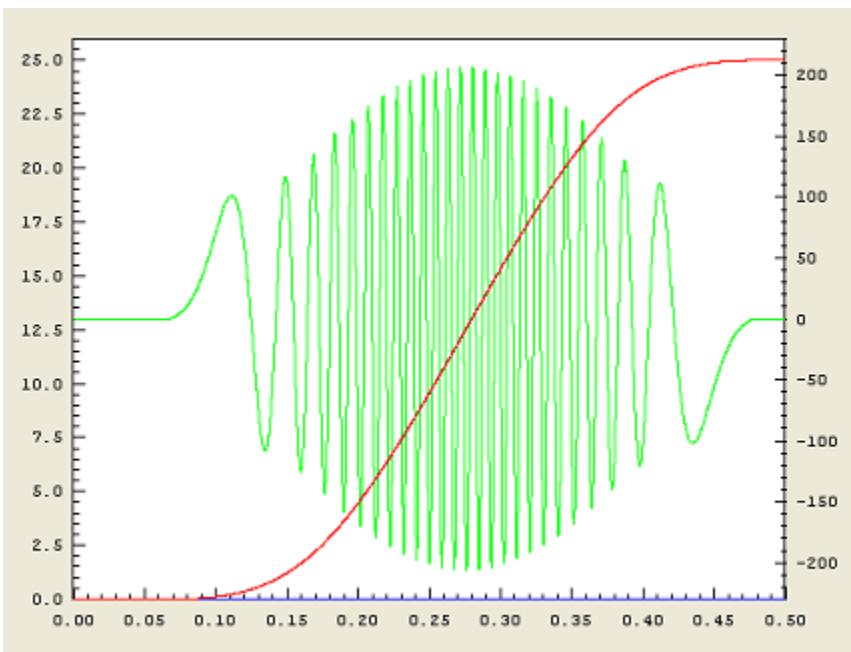
$$lead = \frac{1}{2 \sin(\pi T f_L)}$$

The Following table generated from the above formula may also be used. For most motors the Lead Compensation values will be within the range of 5 - 20.

Freq, Hz	Lead
50	35.37
60	29.47
70	25.26
80	22.11
90	19.65
100	17.69
120	14.74
140	12.63
160	11.06
180	9.83
200	8.85
220	8.04
240	7.37
260	6.81
280	6.32
300	5.90
350	5.06
400	4.43
450	3.94
500	3.55
550	3.23
600	2.96
650	2.74
700	2.54
750	2.38
800	2.23
850	2.10
900	1.99
950	1.88
1000	1.79



This plot above displays a simple 0.5 second motion with no Lead Compensation for a Microstepper Motor. Position axis shown on the primary (left axis) for the red plot has units of cycles. PWM output shown on the secondary (right axis) for the green plot has units of PWM counts. Move parameters are: Vel=200 cycles/sec, Accel=200 cycles/sec², Jerk=10000 cycles/sec³. Note that regardless of velocity PWM amplitude is constant



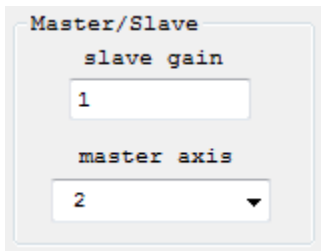
This plot displays the same 0.5 second motion with Lead Compensation = 27.0. All other parameters same as above. Note how PWM amplitude increases with velocity

If motor parameters are unknown, a trial and error approach may be used to find the best lead compensation value. The following procedure may be used:

1. Set Lead Compensation to zero
2. Increase motor speed until a drop in torque is first detected
3. Increase Lead Compensation until normal torque is restored

Setting the Lead Compensation too high should be avoided, as it may cause over current in the motor at medium speeds or voltage distortion due to saturation (clipping).

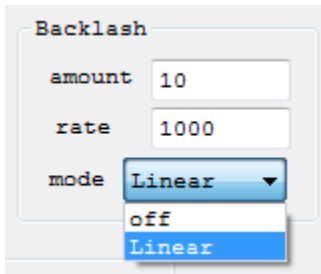
Master/Slave Settings



The Master/Slave Settings dialog box contains two input fields. The first is labeled 'slave gain' and has a text box with the value '1'. The second is labeled 'master axis' and has a dropdown menu with the value '2' selected.

Configures the axis to be slaved to another axis. If slaved when the master axis moves, this axis will be commanded to move by an amount as scaled by the slave gain. If the Slave Gain is negative the slaved axis will move in the opposite direction as the Master. See also Console commands [SlaveGain](#) and [MasterAxis](#). Setting the Master axis as -1 will disable slaving for this axis.

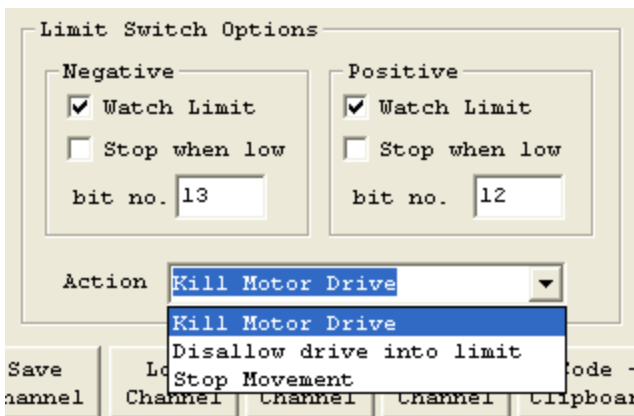
Backlash Settings



The Backlash Settings dialog box contains three input fields. The first is labeled 'amount' and has a text box with the value '10'. The second is labeled 'rate' and has a text box with the value '1000'. The third is labeled 'mode' and has a dropdown menu with 'Linear' selected. The dropdown menu also shows 'off' and 'Linear' as options.

Configures the Backlash Compensation for the axis. To compensate for backlash in an axis, an offset in the commanded position may be applied when moving in the positive direction, and not applied when moving in the negative direction. The amount and rate at which the offset is applied is specified here. See also [BacklashMode](#), [BacklashAmount](#) and [BacklashRate](#) Console commands.

Limit Switch Options



The Limit Switch Options dialog box is divided into two sections: 'Negative' and 'Positive'. In the 'Negative' section, the 'Watch Limit' checkbox is checked, and the 'bit no.' is set to '13'. In the 'Positive' section, the 'Watch Limit' checkbox is checked, and the 'bit no.' is set to '12'. Below these sections is an 'Action' dropdown menu with 'Kill Motor Drive' selected. At the bottom of the dialog are buttons for 'Save', 'Load', and 'Clipboard'.

KMotion has the ability to monitor limit switch inputs for each axis and stop motion when a physical limit switch is detected. The limit switch options allow this feature to be enabled or disabled for each limit (positive or negative), what specific bit to be monitored for each limit, what polarity of the bit indicates contact with the limit, and what action to perform when a limit is detected.

Select *Watch Limit* to enable limit switch monitoring.

Select *Stop when low* to select negative true logic for the limit (motion will be stopped when a low level is detected).

Specify a *bit no.* for which bit is to be monitored for the limit condition. See the [Digital IO Screen](#) for current I/O bit status and a recommended bit assignment for limit switches (bits 12 through 19). If in a particular application it isn't critical to determine which Limit Switch (either positive or negative, or even which axis) the number of digital I/O bits consumed by limit switches may be reduced by "wire ORing" (connecting in parallel) multiple switches together. In this case, the same bit number may be specified more than one place.

The Action drop down specifies what action should be performed when a limit is encountered.

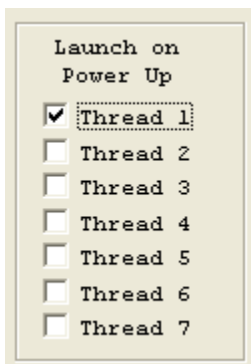
Kill Motor Drive - will completely disable the axis whenever the limit condition is present. Note that it will not be possible to re-enable the axis (and move out of the limit) while the limit condition is still present and this mode remains to be selected.

Disallow drive into limit - will disable the axis whenever the limit condition is present *and* a motion is made into the direction of the limit. This mode will allow the axis to be re-enabled while inside the limit and will allow a move away from the limit.

Stop Movement - this action will keep the axis enabled, but will freeze the commanded position as soon as the limit is detected, and stop any motion trajectory in progress for the axis.

Furthermore, in all the above cases and a limit condition is detected, if the axis belongs to the set of axis in the coordinated group, then any coordinated motion trajectory will be halted.

Launch on Power Up



The launch on power up configuration specifies which User Programs are to be automatically launched on power up for stand alone operation of **KMotion**. See the [C Program Screen](#) for information on how to Edit, Compile, and Download a C program into **KMotion** for execution into one (or more) of the 7 Thread program spaces within **KMotion**.

To configure a program execute on power up, perform the following steps:

1. *Compile and Download* a C Program to a particular Thread Space.
2. Select *Launch on Power Up* for the same Thread.
3. *Flash* the User Memory (see following section).
4. *Disconnect* the Host USB cable
5. *Cycle Power* on the **KMotion**



FLASH

The entire user memory space may be Flashed into nonvolatile memory by depressing the *Flash - User Memory* button. This saves all of the axis configurations, all user program thread spaces, and the user persistent data section. On all subsequent power up resets, **KMotion** will revert to that saved configuration. (note that it is preferred to have the host, or a user program, configure the board before each use rather than relying on the exact state of a KMotion set to a particular state at some point in the past).

To upgrade the system firmware in a **KMotion** use the *Flash - New Version* button. The user will be prompted to select a **DSPKMotion.out** COFF file from within the **KMotion** Install Directory to download and Flash. Note that all user programs and data will be deleted from **KMotion** when loading a new version.

After the firmware has been flashed it is necessary to re-boot the **KMotion** in order for the new firmware to become active.

It is important that the **<Install Directory>\DSP_KMotion\DSPKMotion.out** file match the firmware that is flashed into **KMotion**. User C programs are *Linked* using this file to make calls and to access data located within the **KMotion** firmware. Whenever a user program is compiled and linked using this file, the timestamp of this file is compared against the timestamp of the executing firmware (if a **KMotion** is currently connected). If the timestamps differ, the following message will be displayed, and it is not recommended to continue. The "Version" Console Script Command may also be used to check the firmware version.

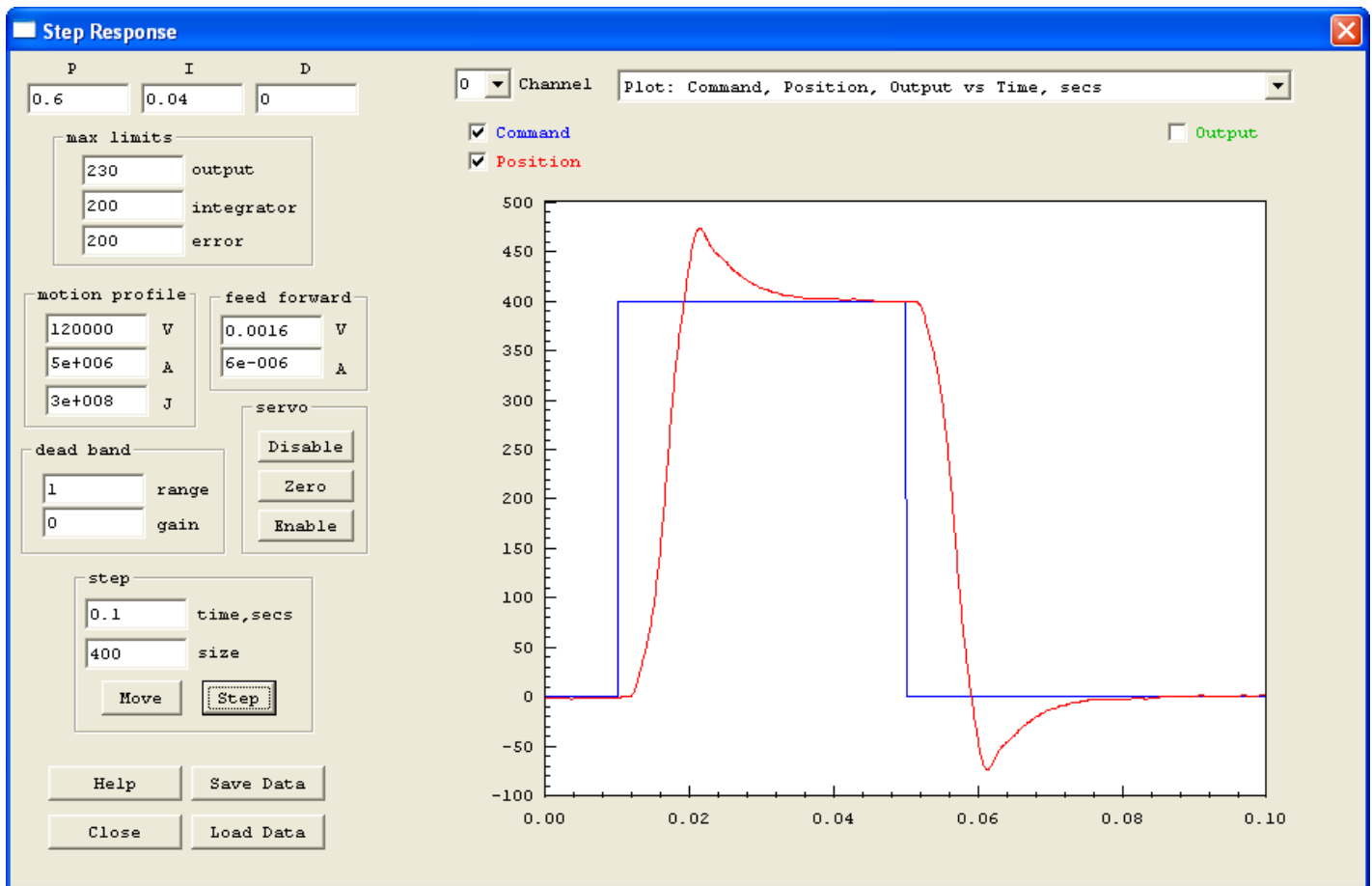


In all cases while flashing firmware or user programs the process should not be interrupted or a corrupted flash image may result which renders the board un-bootable. However if this occurs the

Flash Recovery mode may be used to recover from the situation. To perform the recovery, press the *Flash Recovery* button and follow the dialog prompts to:

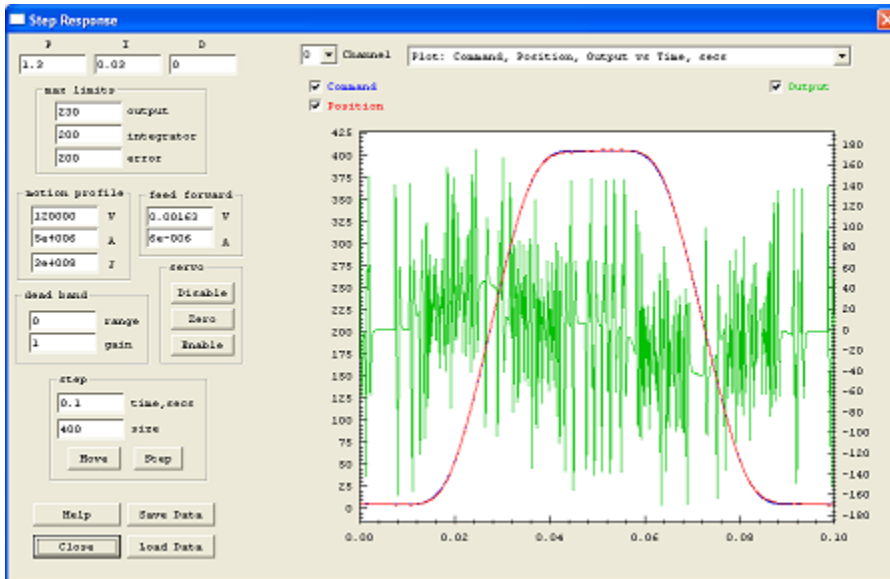
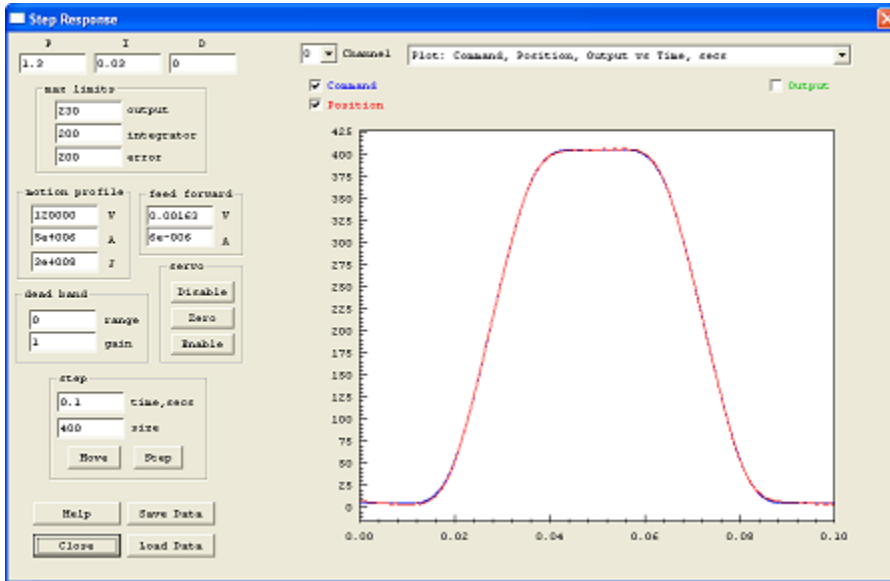
1. Select the firmware file to boot
2. Turn off **KMotion**
3. Turn on KMotion
4. After **KMotion** boots, Flash the New Version

Step Response Screen

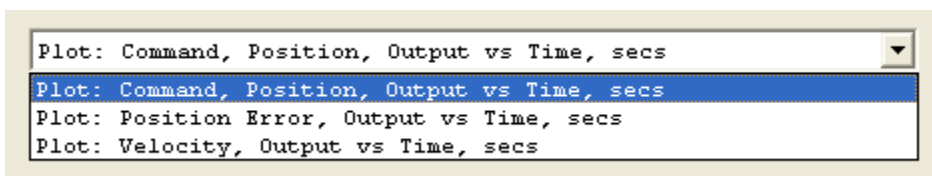


(Click on above image to jump to relative topic)

The **Step Response Screen** allows changes to system tuning parameters and allows measurement and graphs of the system's time response for either a Move Profile or a Step Function. The graph shown above is of an applied step function of 400 counts. The graphs shown below are of a profiled move (and back) of 400 counts. The first has the Output drive hidden and the second has the Output drive displayed. Click on the graphs for a larger view. Note that the Output drive signal contains large spikes. This is the result of quantization error in the measured position. Quantization error in the measured position makes it appear to the system as if there was no motion, and then suddenly as if there was a relatively quick motion of one count in a single servo sample cycle. This is a [non-linear effect](#). In some cases these "spikes" may exceed the output range causing saturation a still further non-linear effect. A low pass filter may be used to "smooth" the output, see the [IIR Filter Screen](#), but has limits. Attempting too much "smoothing" by setting a lower frequency will eventually have an effect on the performance of the system, reducing the [phase margin](#). Normally, the cutoff frequency of the low pass filter should be significantly larger than the system bandwidth.



There are three basic time domain plot types that may be selected from the drop down list, which are shown below.



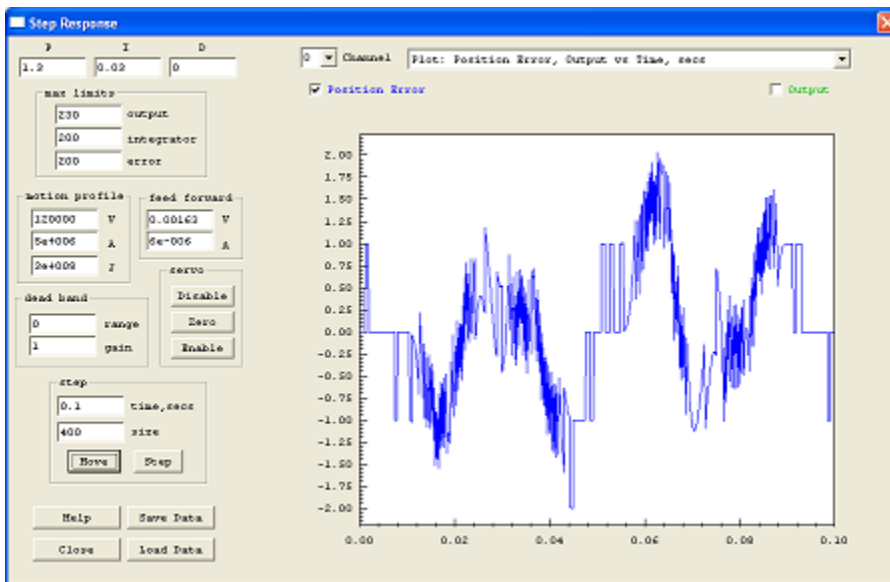
They consist of either:

1. Commanded Position, Measured Position, and Motor Output
2. Position Error and Motor Output
3. Commanded Velocity, Measured Velocity, and Motor Output

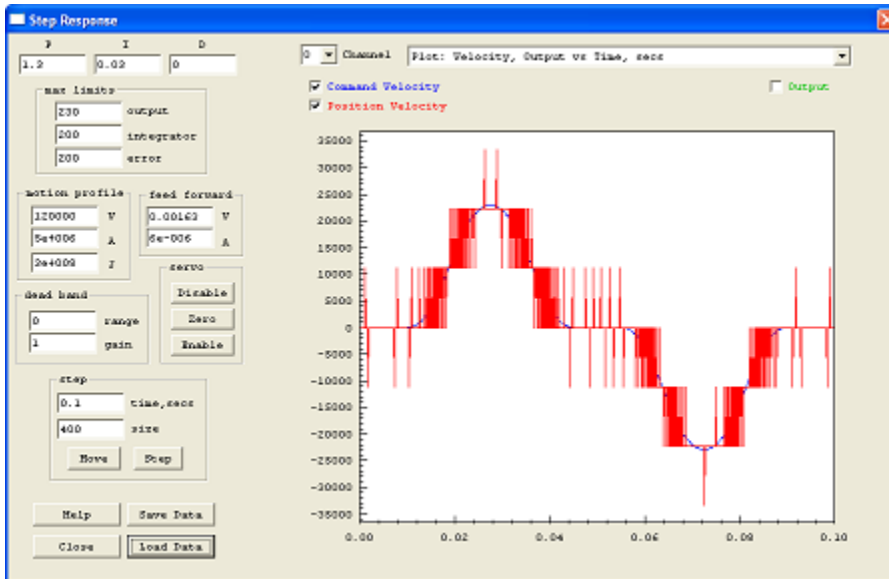
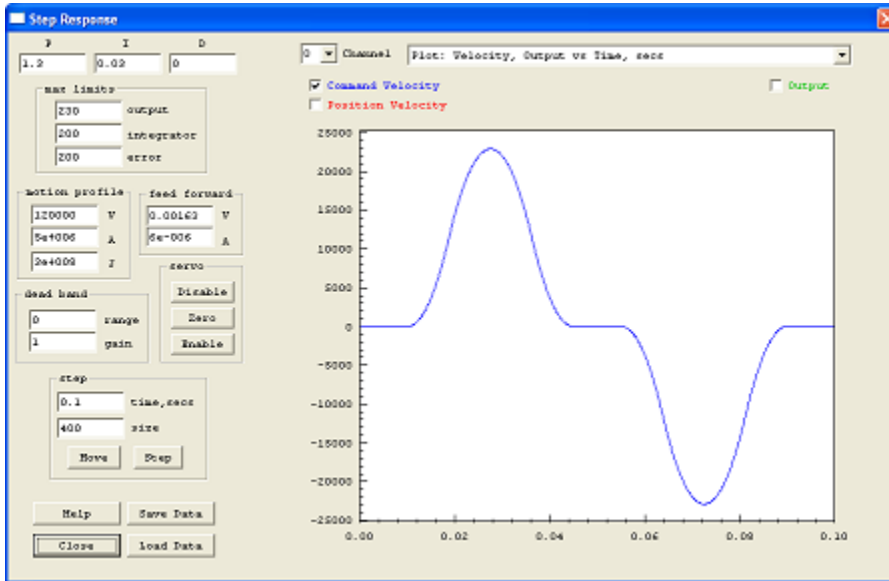
For all three plot types the Motor Output is always displayed as a secondary Y axis on the right side of the graph. The other plotted values are on the primary left Y axis. The X axis is always time in seconds. After a particular plot type has been selected, each individual plot variable may be displayed or hidden by selecting the checkbox with the corresponding name (and color) of the variable.

Any portion of the graph may be zoomed by left-click dragging across the graph. Simply select the area of interest. Right clicking on the graph will bring up a context menu that allows zooming out completely or to the previous zoom level.

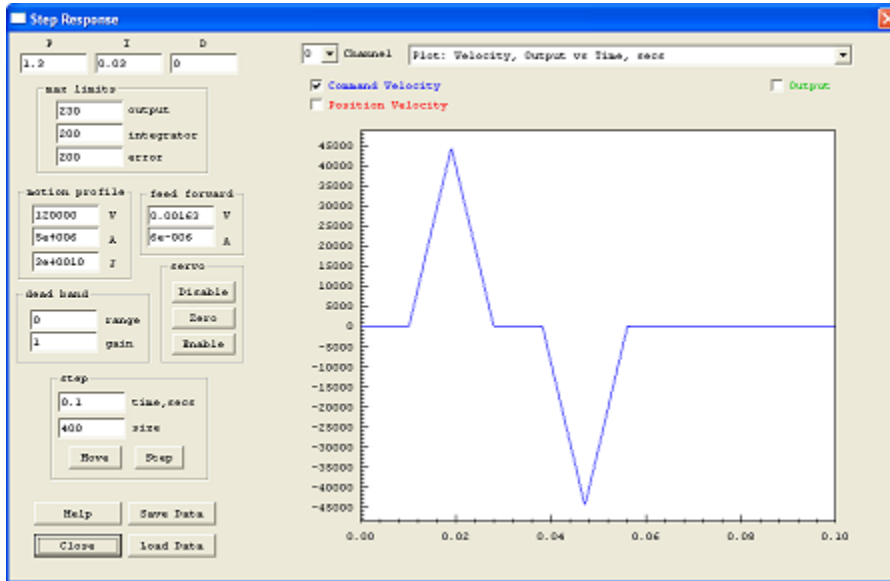
Below is an example of a graph of *Position Error* (for the same 400 count move shown above). Position Error is defined as Measured Position - Commanded Position. The same data as that is plotted in the Command/Position plots is used, however instead of plotting both values, the difference is plotted. Note that because the Measured Position is quantized to integer encoder counts, the quantization effect is also observed in the Position Error.



The third type of plot displays the Velocity of the Commanded and/or Measured Position. Velocity units are *Position Units* per second. When a Move is commanded, a motion profile is computed which achieves the motion in the shortest time without exceeding the maximum allowed velocity, acceleration, or jerk. Because the Command is a theoretical profile computed using floating point arithmetic, it is very smooth. The blue graph immediately below shows such a plot. In a velocity graph, *slope* in the graph represents acceleration. In this case a relatively low value specified for maximum jerk causes the changes in slope to be gradual. The second plot below is the same data but with the *Measured* velocity displayed along with the Commanded velocity. Because of encoder resolution limitations, measured velocity calculated using a simple position difference per sample period tends to be highly quantized as shown. In this example even at our peak velocity at ~ 23,000 position counts per second this results in a maximum of only 3 position counts per servo sample period.



The velocity graph below, shows the effect of setting the maximum allowed jerk to a very large value (100X higher than the graph above). Note how the slope of the velocity changes abruptly which represents a high rate of change of acceleration (jerk).



Tuning Parameters - PID

P	I	D
1.3	0.03	0

The PID (proportional, integral, and derivative) gains set the amount of feedback of the error itself (proportional), the integration of the error (integral), and the derivative of the position (derivative) that is applied to the output. Also see the

[KMotion Servo Flow Diagram.](#)

The units of the proportional gain are in Output Units/Position Units. For example if the Position Units are in encoder counts, and the Output Units are in [PWM counts](#), then a gain of 10.0 would apply an output drive of 10 PWM for an error of 1 encoder count.

The units of the integral gain are in Output Units/Position Units per *Servo Sample Time*. **KMotion's Servo Sample Time** is fixed at 90µs. An integrator basically sums the position error every servo sample. For example, with an integral gain of 10, and an error of 1 encoder count for 5 servo samples, an output drive of 50 PWM counts would be applied. Integrator gain is normally used to achieve high accuracy. This is because even a very small error will eventually integrate to a large enough value for there to be a corrective action. In fact, having any integrator gain at all guarantees a steady state error (average error) of zero. This effect also guarantees that there will always be some overshoot in response to a step function, otherwise the average error could not be equal to zero.

The units of the derivative gain are in Output Units/Position Units x *Servo Sample Time*. The derivative term is simply the change in position from one servo sample to the next. For example, with a derivative gain of 10, and a position change of 1 encoder count from the previous servo sample, an output drive of -10 PWM counts would be applied. The negative sign shows that the output is applied in a manner to oppose motion. Derivative gain has the effect of applying damping, which is a force proportional and opposite to the current velocity. Although derivative gain is often used successfully in a control system, consider using a lead/lag filter which performs in a similar manner, but doesn't have the undesirable feature of increasing gain at high frequencies.

max limits	
230	output
200	integrator
200	error

Tuning Parameters - max limits

KMotion's *max limits* allow several points in the [Servo Flow Diagram](#) to be clamped, or limited to a specified range. The limits in the flow



diagram are shown as a clamp symbol . This capability is often useful in controlling how the system responds to extreme situations.

Maximum *output* limit is used to limit the maximum applied value, in counts, to the output drive. The output drive may be either one of the on-board [PWM outputs](#) or a [DAC value](#) that drives an external amplifier.

Maximum *integrator* limit is used to restrict the maximum value of the integrator. This effect is often used to avoid an effect referred to as integrator "*wind up*". Without any integrator limit, consider the case where somehow a substantial error is maintained for a significant period of time. For example turning a motor shaft by hand for several seconds. During this time the integrator would ramp up to an extremely large value. When the motor shaft was released, it would accelerate at maximum and overshoot the target by a huge amount until the integrator could ramp back down to a reasonable value. This often results in a servo slamming into a limit. The maximum integrator limit prevents this from occurring. Often the main purpose for using an integrator is to overcome static friction in order to reduce the final error to zero. This usually requires only a small fraction of total output range. In almost all cases it is of no value to allow the integrator to exceed the maximum output value.

Maximum *error* limits the maximum value allowed to pass through the servo compensator. The units are the same as position units. Typically, when a servo loop is operating normally, its following error is a small value. When some extreme even occurs, such as a sudden large step command, or possibly a large disturbance the error may become very large. In some cases there may be benefit to limiting the error to a reasonable value.

Tuning Parameters - Motion Profile

motion profile	
120000	V
5e+006	A
3e+008	J

The Motion Profile parameters set the maximum allowed velocity (in position units per second), the maximum allowed acceleration (in position units per second²), and the maximum allowed jerk (in position units per second³). These parameters will be utilized for any independent (non coordinated motion) move command for the axis. The acceleration and jerk also apply to jog commands (move at continuous velocity) for the axis.

feed forward	
0.00163	V
6e-006	A

Tuning Parameters - Feed Forward

KMotion's *Feed Forward* may often be used to dramatically reduce the following error in a system. See the [Servo Flow Diagram](#) to see precisely how it is implemented. The idea behind feed forward is to observe the velocity and acceleration of the command signal and anticipate a required output and to apply it without waiting for an error to develop.

Most motion systems are constructed in manner where some sort of motor force is used to accelerate a mass. In these cases whenever an acceleration is required a force proportional to the acceleration will be required to achieve it. Acceleration feed forward may be used to reduce the amount that the feedback loop must correct. In fact, proper feed forward reduces the requirement on the feedback from the total force required to accelerate the mass, to only the variation in the force required to accelerate the mass.

Similarly most servo systems require some amount of force that is proportional to velocity simply to maintain a constant velocity. This might be due to viscous friction, or possibly motor back emf (electro motive force). In any case velocity feed forward may be used to reduce the demands of the feedback loop resulting in smaller following error.

The normal procedure to optimize feed forward is to select plot type - position error, and measure moves using the *Move Command* (Step functions should *not* be used as step functions are instantaneous changes in position that represent infinite velocity and acceleration).

Note that in the [Servo Flow Diagram](#) the feed forward is injected *before* the final [IIR Filter](#). This allows any feed forward waveforms to be conditioned by this filter. Feed forward pulses may be relatively sharp pulses to make rapid accelerations that may often tend to disturb a mechanical resonance in the system. Usually a system with a sharp resonance will benefit from a notch filter to improve the stability and performance of the servo loop. By placing the notch filter as the last filter in the servo loop, the feed forward waveform will also pass through this filter and the result is that the feed forward will cause less excitation of the mechanism than it would otherwise..

Tuning Parameters - Dead Band

dead band

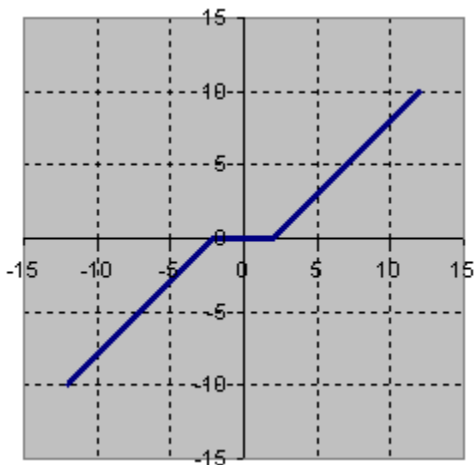
0

range

1

gain

Dead band is used to apply a different gain to the region near zero than the rest of the region. Usually either zero gain or a gain much less than 1 is used within the dead band range. See the [Servo Flow Diagram](#) for the exact location of where the dead band is inserted. Dead band is a means of introducing "slop" into a system. This usually results in less accuracy and performance, but may reduce or eliminate limit cycle oscillations while resting at the target position.



The values shown (range = 0, gain = 1) are used to defeat any dead band. The chart shows the resulting input/output for range = 2, gain = 0. The slope of the graph is always 1 outside of the specified +/- range, and the specified gain with the +/- range.

Measurement

To perform a measurement and display the response, select the time duration to gather data, and the move or step size to perform, and press either the Move or Step buttons. If the axis is currently enabled, it will be disabled, all parameters from all screens will be downloaded, the axis will be enabled, the move or step will be performed while the data is gathered, the data will then be uploaded and plotted.

A *Move* will hold position for a short time, perform a motion of the specified amount from the current location, pause for a short time, and then a second motion back to the original location.

A *Step* will hold position for a short time, perform a step of the specified amount from the current location, pause for a short time, and then a second step back to the original location.

The maximum time that data may be collected is 3.5 seconds (3.5 seconds / 90μs = 38,888 data points). Note that collecting data at this rate allows zooming while still maintaining high resolution.

Axis Control

The Axis Control buttons are present to conveniently disable (Kill), Zero, or Enable an axis. If the axis becomes unstable (possible due to a gain setting too high), the Kill button may be used to disable the axis, the gain might then be reduced, and then the axis may be enabled. The Enable button downloads all parameters from all screens before enabling the axis in the same manner as the [Measurement](#) buttons described above.

Note for brushless output modes that commutate the motor based on the current position, Zeroing the position may adversely affect the commutation.

Save/Load Data

The Save/Load Data buttons allow the captured Step Plot to be saved to a text file and re-loaded at a later time. The text file format also allows the data to be imported into some other program for display or analysis. The file format consists of one line of header followed by one line of 5 comma separated values, one line for each sample. The values are:

1. Sample Number
2. Time, Seconds
3. Command
4. Position
5. Output

Example of data file follows:

Sample,Time,Command,Position,Output

0,0,5,5,-0.3301919

1,9e-005,5,5,-0.3300979

2,0.00018,5,5,-0.3300258

3,0.00027,5,5,-0.3299877

4,0.00036,5,5,-0.3299999

5,0.00045,5,5,-0.3300253

6,0.00054,5,5,-0.3300359

7,0.00063,5,5,-0.3300304

8,0.00072,5,5,-0.3300199

9,0.00081,5,5,-0.3300156

10,0.0009,5,5,-0.3300157

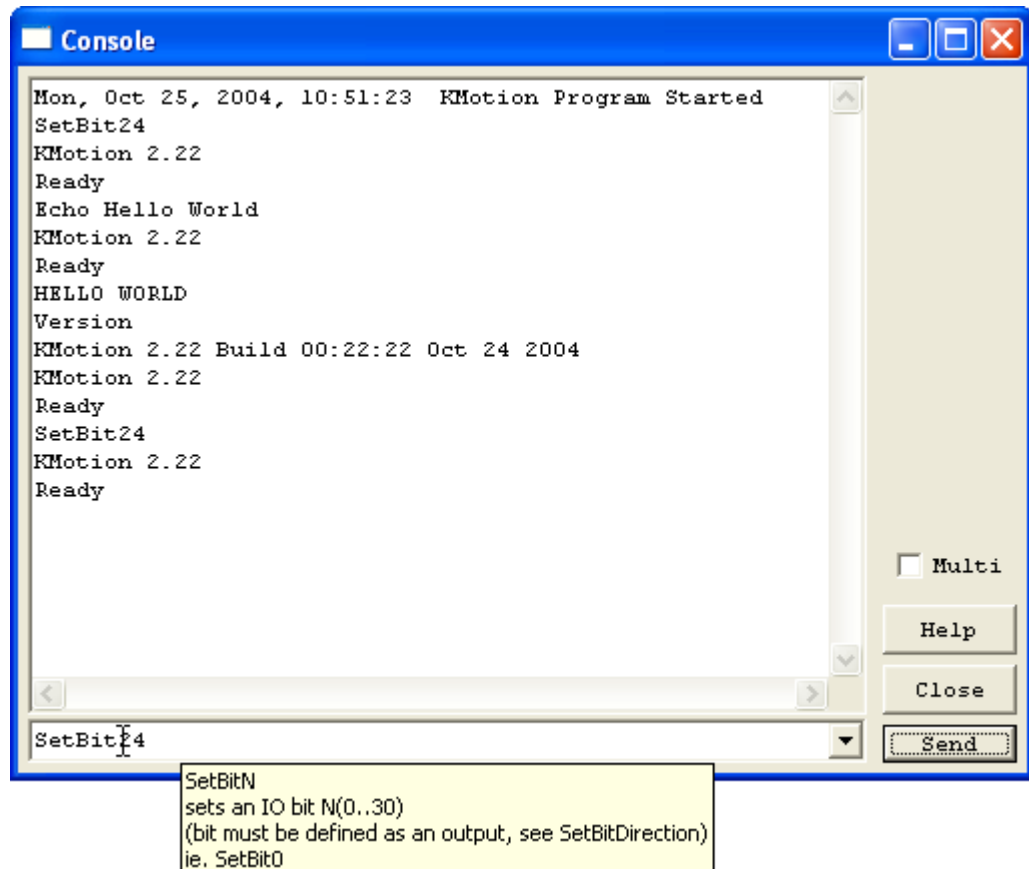
62

-
-
-

Console Screen

Commands (alphabetical):

[3PH<N>=<M> <A>](#)
[4PH<N>=<M> <A>](#)
[Accel <N>=<A>](#)
[ADC<N>](#)
[Arc <XC> <YC> <RX> <RY>](#)
[<θ0> <dθ> <Z0> <A0> <B0> <C0>](#)
[<Z1> <A1> <B1> <C1>](#)
[<a> <c> <d> <tF>](#)
[ArcXZ <XC> <ZC> <RX> <RZ>](#)
[<θ0> <dθ> <Y0> <A0> <B0> <C0>](#)
[<Y1> <A1> <B1> <C1>](#)
[<a> <c> <d> <tF>](#)
[ArcYZ <YC> <ZC> <RY> <RZ>](#)
[<θ0> <dθ> <X0> <A0> <B0> <C0>](#)
[<X1> <A1> <B1> <C1>](#)
[<a> <c> <d> <tF>](#)
[ArcHex <XC> <YC> <RX> <RY>](#)
[<θ0> <dθ> <Z0> <A0> <B0> <C0>](#)
[<Z1> <A1> <B1> <C1>](#)
[<a> <c> <d> <tF>](#)
[ArcHexXZ <XC> <ZC> <RX> <RZ>](#)
[<θ0> <dθ> <Y0> <A0> <B0> <C0>](#)
[<Y1> <A1> <B1> <C1>](#)
[<a> <c> <d> <tF>](#)
[ArcHexYZ <YC> <ZC> <RY> <RZ>](#)
[<θ0> <dθ> <X0> <A0> <B0> <C0>](#)
[<X1> <A1> <B1> <C1>](#)
[<a> <c> <d> <tF>](#)
[CheckDone<N>](#)
[CheckDoneBuf](#)
[CheckDoneGather](#)
[CheckDoneXYZA](#)
[ClearBit<N>](#)
[ClearBitBuf<N>](#)
[ClearFlashImage](#)
[CommutationOffset<N>=<X>](#)
[D<N>=<M>](#)
[DAC<N> <M>](#)
[DeadBandGain<N>=<M>](#)
[DeadBandRange<N>=<M>](#)
[DefineCS<X> <Y> <Z> <A>](#)
[<C>](#)
[Dest<N>=<M>](#)
[DisableAxis<N>](#)
[Echo <S>](#)
[EnableAxis<N>](#)
[EnableAxisDest<N> <M>](#)
[Enabled<N>](#)
[EntryPoint<N> <H>](#)
[ExecBuf](#)
[ExecTime](#)
[Execute<N>](#)
[FFAccel<N>=<M>](#)
[FFVel<N>=<M>](#)
[Flash](#)
[GatherMove<N> <M> <L>](#)
[GatherStep<N> <M> <L>](#)
[GetBitDirection<N>](#)
[GetGather <N>](#)
[GetGatherDec<N>](#)
[GetGatherHex<N> <M>](#)
[GetInject<N> <M>](#)
[GetPersistDec<N>](#)
[GetPersistHex<N>](#)
[GetStatus](#)
[I<N>=<M>](#)
[IIR<N> <M>=<A1> <A2> <B0> <B1>](#)

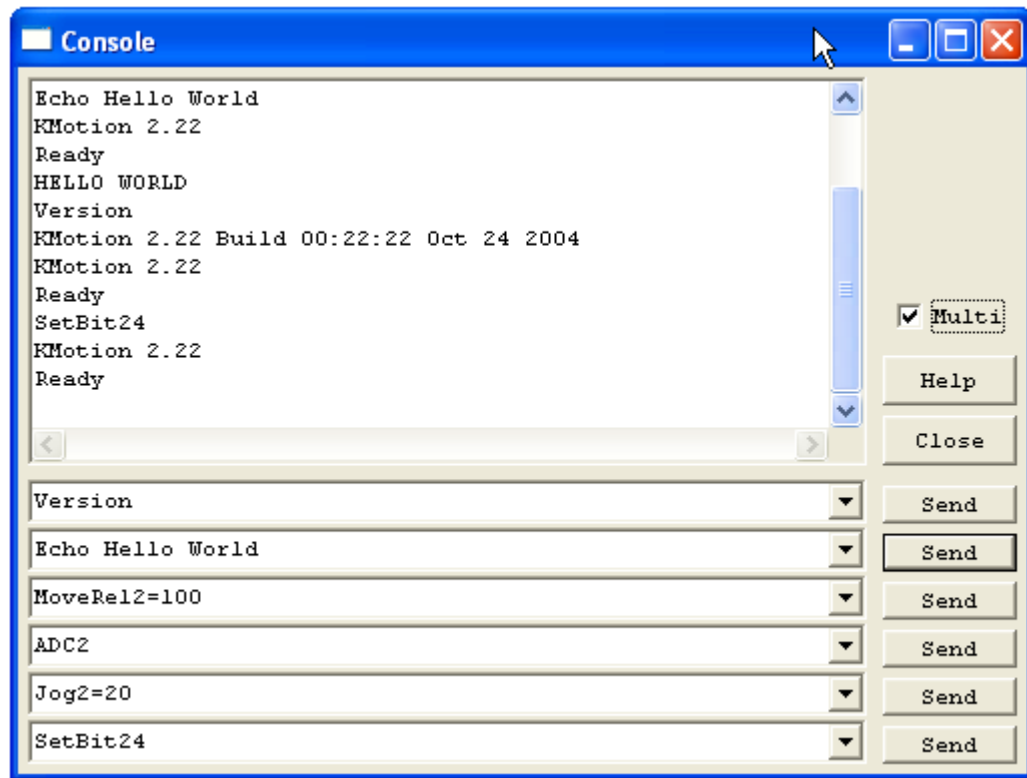


The **Console Screen** displays messages from the DSP and the PC. The Console window retains the last 1000 lines of text. After more than 1000 lines are displayed the earliest messages scroll off into a permanent text file (**LogFile.txt**) in the **KMotion\Data** subdirectory.

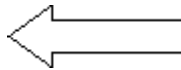
To Send a command to the DSP enter the text string in the bottom command cell and press the **Send** button.

Selecting the ☒ **Multi** Check box changes from a single command line to multiple command lines, see below. This allows several commands to be entered and then easily sent with a single push button.

[<B2>](#)
[Inject<N> <F> <A>](#)
[InputChan<M> <N>=<C>](#)
[InputGain<M> <N>=<G>](#)
[InputMode<N>=<M>](#)
[InputOffset<M> <N>=<O>](#)
[InvDistPerCycle<N>=<X>](#)
[Jerk<N>=<J>](#)
[Jog<N>=<V>](#)
[Kill<N>](#)
[Lead<N>=<M>](#)
[LimitSwitch<N>=<H>](#)
[Linear <X0> <Y0> <Z0> <A0> <B0>
\[<C0>\]\(#\)
\[<X1> <Y1> <Z1> <A1> <B1> <C1>\]\(#\)
\[<a> <c> <d> <tF>\]\(#\)
\[LinearHex <X0> <Y0> <Z0> <A0>
\\[<B0> <C0>\\]\\(#\\)
\\[<X1> <Y1> <Z1> <A1> <B1> <C1>\\]\\(#\\)
\\[<a> <c> <d> <tF>\\]\\(#\\)
\\[LoadData <H> <N>\\]\\(#\\)
\\[LoadFlash<H> <N>\\]\\(#\\)
\\[MaxErr<N>=<M>\\]\\(#\\)
\\[MaxFollowingError<N>=<M>\\]\\(#\\)
\\[MaxI<N> <M>\\]\\(#\\)
\\[MaxOutput<N>=<M>\\]\\(#\\)
\\[Move<N>=<M>\\]\\(#\\)
\\[MoveAtVel<N>=<M> <V>\\]\\(#\\)
\\[MoveRel<N>=<M>\\]\\(#\\)
\\[MoveRelAtVel<N>=<M> <V>\\]\\(#\\)
\\[MoveXYZA <X> <Y> <Z> <A>\\]\\(#\\)
\\[OpenBuf\\]\\(#\\)
\\[OutputChan<M> <N>=<C>\\]\\(#\\)
\\[OutputMode<N>=<M>\\]\\(#\\)
\\[P<N>=<M>\\]\\(#\\)
\\[Pos<N>=<P>\\]\\(#\\)
\\[ProgFlashImage\\]\\(#\\)
\\[PWM<N>=<M>\\]\\(#\\)
\\[PWMR<N>=<M>\\]\\(#\\)
\\[ReadBit<N>\\]\\(#\\)
\\[Reboot!\\]\\(#\\)
\\[SetBit<N>\\]\\(#\\)
\\[SetBitBuf<N>\\]\\(#\\)
\\[SetBitDirection<N>=<M>\\]\\(#\\)
\\[SetGatherDec <N> <M>\\]\\(#\\)
\\[SetGatherHex<N> <M>\\]\\(#\\)
\\[SetPersistDec <O> <D>\\]\\(#\\)
\\[SetPersistHex <O> <H>\\]\\(#\\)
\\[SetStartupThread<N> <M>\\]\\(#\\)
\\[SetStateBit<N>=<M>\\]\\(#\\)
\\[SetStateBitBuf<N>=<M>\\]\\(#\\)
\\[StepperAmplitude<N>=<M>\\]\\(#\\)
\\[Vel<N>=<V>\\]\\(#\\)
\\[Version\\]\\(#\\)
\\[Zero<N>\\]\\(#\\)\]\(#\)](#)



See the alphabetical list for available commands.



Or see commands grouped by category [here](#).

Axis Status Screen

The screenshot shows a window titled "Axis" with a close button (X) in the top right corner. The window contains a table with the following columns: Dest, Position, Enable, Modes, and Done. The table lists 8 axes (#0 to #7). Axes #0, #1, and #2 are enabled and have "Encoder Step Dir" mode. Axes #3, #4, #5, #6, and #7 are disabled and have "Encoder No Output" mode. The "Done" column has checkboxes for each axis, with #0, #1, and #2 checked.

Dest	Position	Enable	Modes	Done
#0	0.00	#0	0 <input checked="" type="checkbox"/>	Encoder Step Dir <input checked="" type="checkbox"/>
#1	0.00	#1	0 <input checked="" type="checkbox"/>	Encoder Step Dir <input checked="" type="checkbox"/>
#2	0.00	#2	0 <input checked="" type="checkbox"/>	Encoder Step Dir <input checked="" type="checkbox"/>
#3	0.00	#3	0 <input type="checkbox"/>	Encoder No Output <input type="checkbox"/>
#4	0.00	#4	0 <input type="checkbox"/>	Encoder No Output <input type="checkbox"/>
#5	0.00	#5	0 <input type="checkbox"/>	Encoder No Output <input type="checkbox"/>
#6	0.00	#6	0 <input type="checkbox"/>	Encoder No Output <input type="checkbox"/>
#7	0.00	#7	0 <input type="checkbox"/>	Encoder No Output <input type="checkbox"/>

At the bottom of the window are two buttons: "Help" and "Close".

The **Axis Screen** displays Axis Status including:

- Current Commanded Destination
- Current Measured Position
- Whether the axis is currently enabled
- Input Mode
- Output Mode
- Done Status (no independent Move or Jog in progress)
- Current Bar Graph (if current sensing is available for the axis)

Clicking on Enable will enable or disable the axis without downloading any parameter settings to KFLOP.



Eight-Axis, DSP/FPGA-based Motion Controller

DynoMotion's KFLOP card combines a 1.2 GFLOP DSP (TMS320C6722), FPGA, USB, and a PC-based development environment to create a versatile and programmable motion solution. Designed for up to eight-axes control, KFLOP provides advanced control for torque, speed, and position for any mix of stepper, DC brushless, and DC brush motors. KFLOP uses flash memory to store and run multiple-thread compiled C code on a 1.2 GFLOP processor with native 64-bit floating point support for stand-alone operation. A PC connected with a USB cable can be used for control and monitoring.

The included PC-based integrated development environment combines configuration, status, programming, and advanced diagnostic and tuning tools such as Bode plots and signal filtering. GCode support allows coordinated moves between axes. Libraries for controlling the KFLOP card via Visual C++ and Visual Basic are included, as well as a free C compiler. Thread-safe operation allows the IDE to be used in conjunction with a user application for control and debugging.

The KFLOP packs a lot of IO into its 5.0 x 3.5 in package. KFLOP offers 45 Bi-directional I/O bits, shared between dedicated IO and user-defined I/O.

<http://dynamotion.com>, Calabasas, CA. [sales@dynamotion.com]

THANK YOU!

for purchasing KFLOP

1.2 GFLOP Motion Controller

by Dynomotion, Inc.

www.dynomotion.com

Email: support@dynomotion.com

Please download and install the latest software from

<http://dynomotion.com/Software/Download.html>

The installation will create a Windows™ Start button link to further help and to the **KMotion** Setup and Tuning Application.

QuickStart

Remove the KFLOP Board from the anti-static packaging in a static safe environment.

Note: immediately before touching any electronic component, always discharge any static electricity you may have by touching an earth ground, such as the metal chassis of a PC.

KFLOP may operate powered from the USB by inserting J3. Total power must be < 0.5A for USB powered operation. Otherwise remove J3 and connect a +5V power supply to the **KMotion** 4 pin Molex connector. This connector is the same Pinout as a PC Disk Drive power connector. A PC power supply makes an excellent low cost power source for **KFLOP**. The +12V is not required for operation, it is routed internally through the board to several of the connectors. See the on-line help section titled *Hardware/Connector Description* for more information.

Note: KFLOP may at times draw as little as 0.25 Amps from the +5V supply. Some PC power supplies will not function without a minimum load. In these cases an appropriate power resistor (~10 ohm 5 Watt) should be added across the +5V. Additionally, most ATX power supplies require pins 14 and 15 on the main 20 pin power connector to be shorted.

Note: it is NOT recommended to use the same power supply that is powering your PC and Hard drives to power the KFLOP. Motor noise and power surges have the possibility to cause damage or loss of data within the PC.

Connect the USB or turn on the +5V supply. Two green LEDs should blink rapidly for a few seconds (**KFLOP** is checking if the Host is requesting a Flash Recovery during this time), and then the LEDs should remain steady.

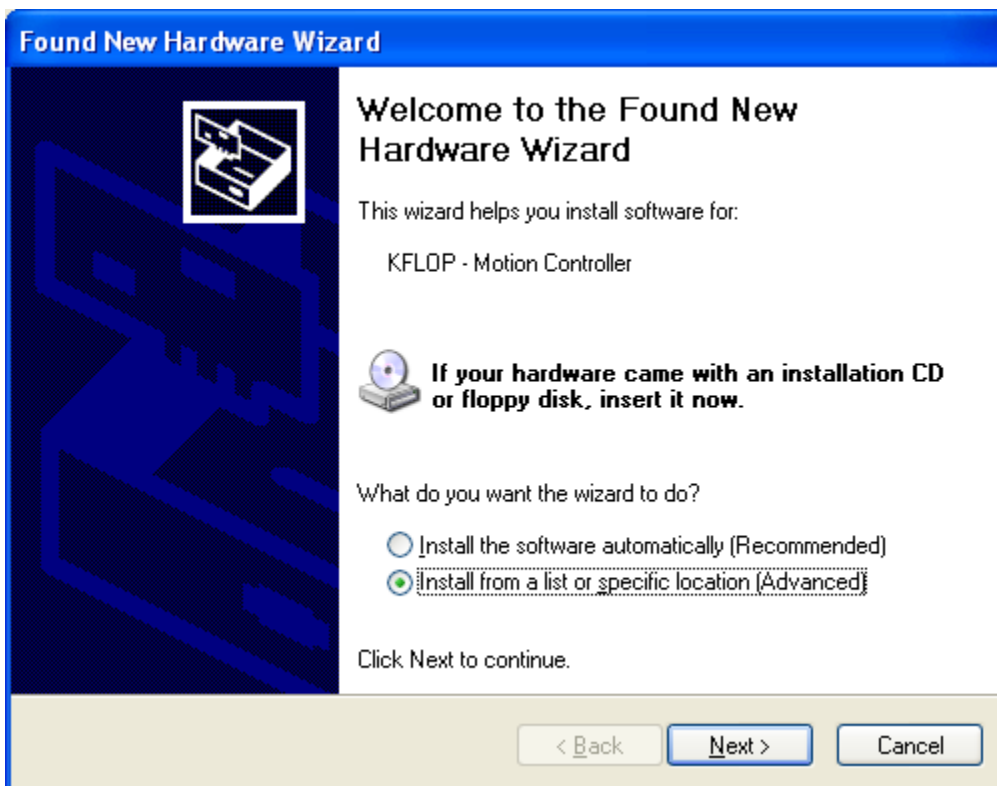
Now connect a USB cable from **KFLOP** to your PC and follow the USB Driver instructions below.

USB Installation

The first time the **KFLOP** board is connected to a computer's USB port this message should be displayed near the Windows™ Task Bar.



Shortly thereafter, the New Hardware Wizard Should appear.

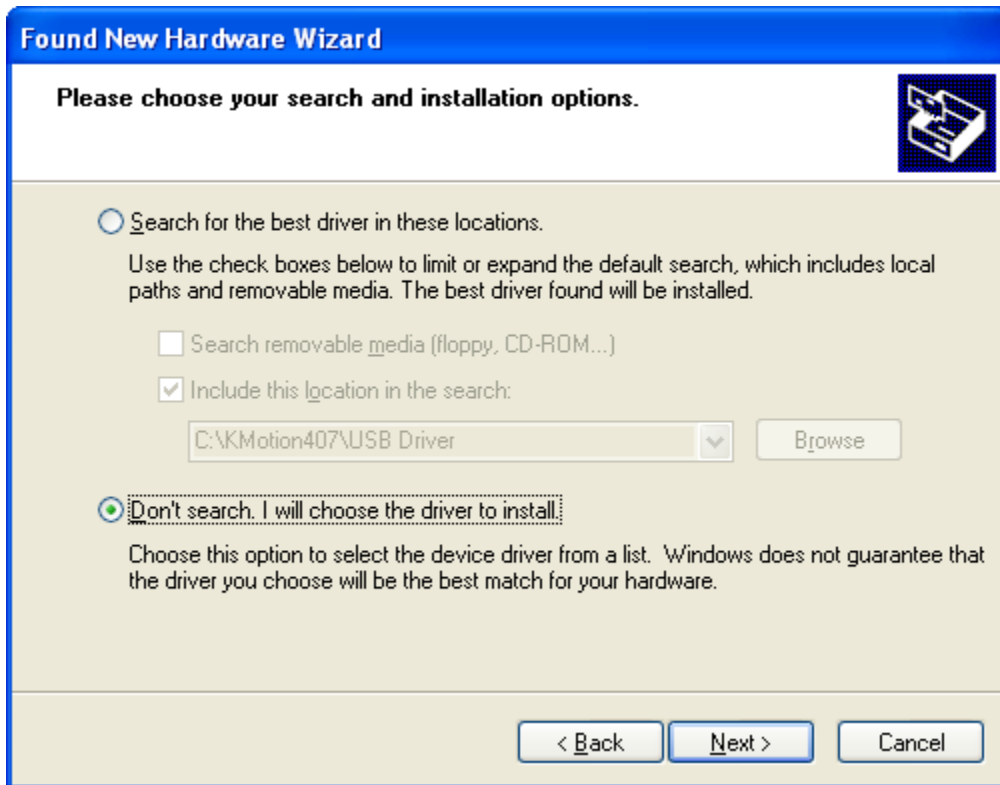


Select "**Install from a specific location (Advanced)**" and select **Next**.

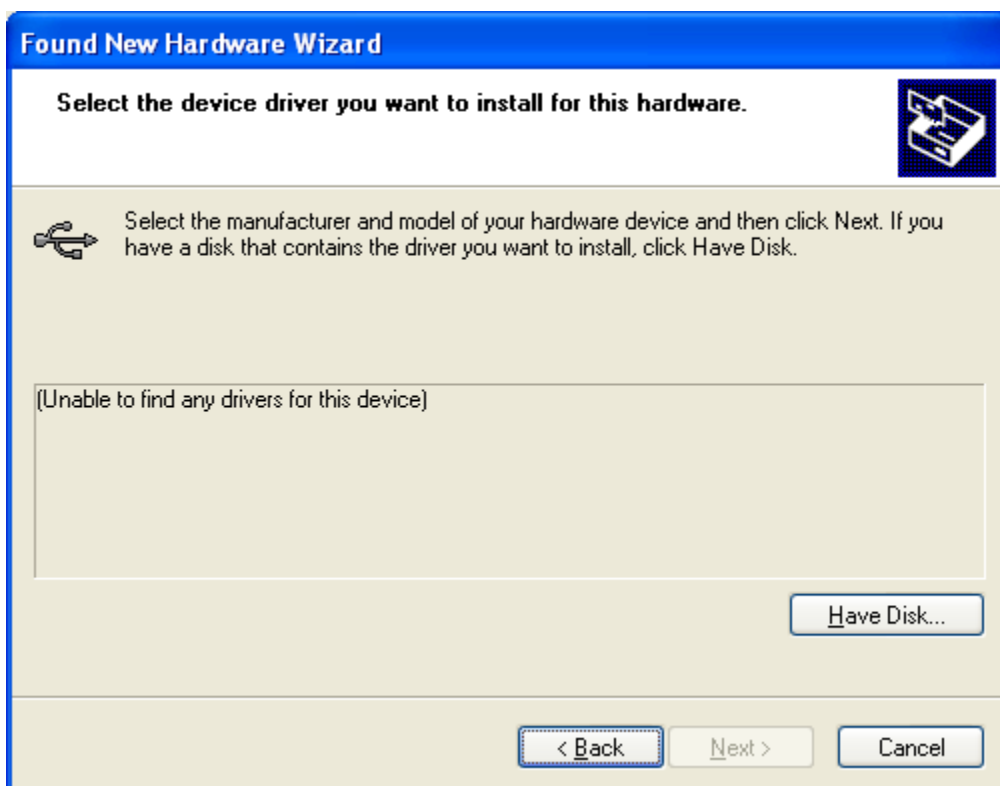
If you haven't already, download and install the complete **KFLOP** Software including drivers available at:

<http://dynamotion.com/Software/Download.html>

Select: "**Don't search. I will choose the driver to install**" and click **Next**



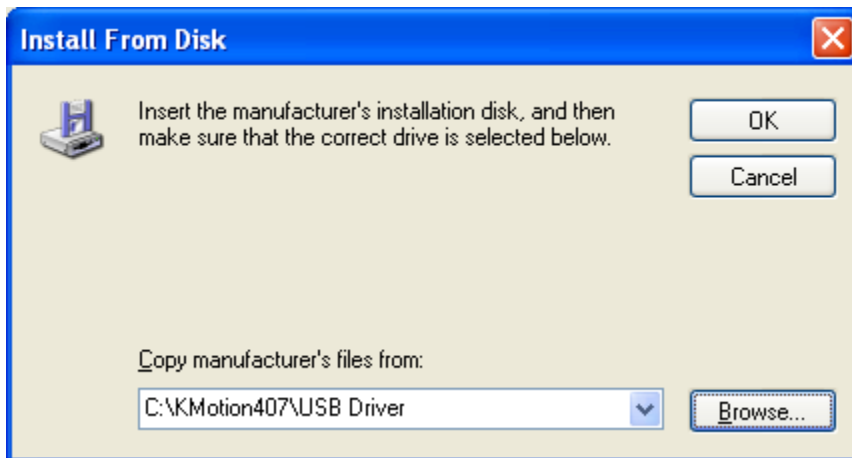
Select: "Have Disk..."



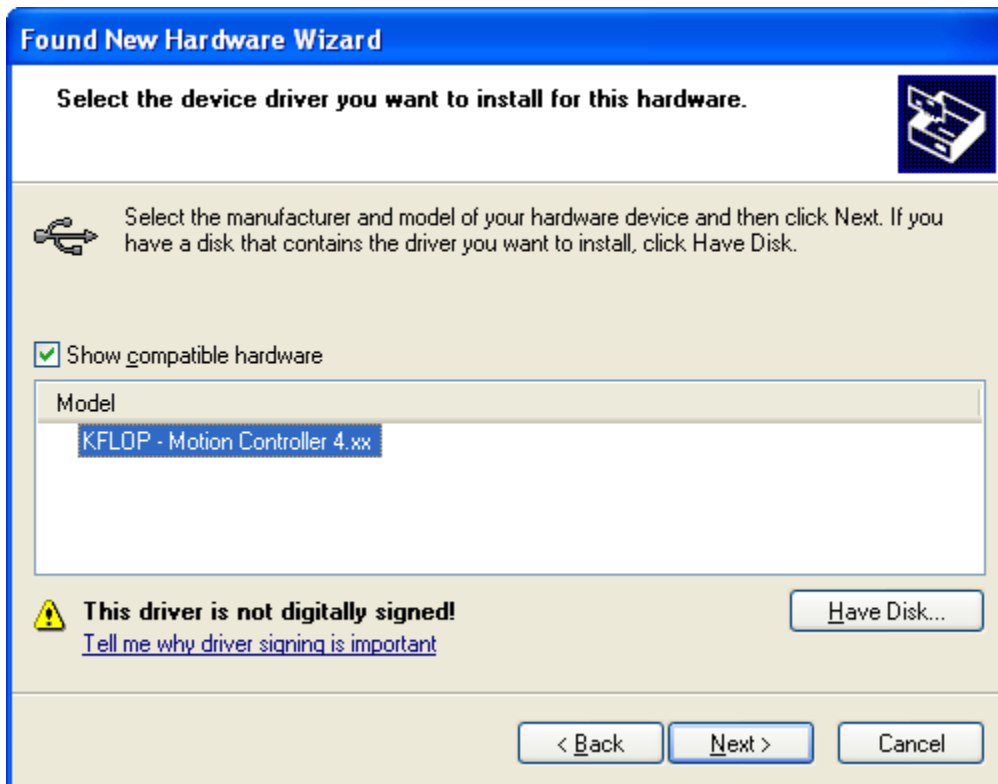
Browse to within the subdirectory where you selected the **KFLOP** Software to be installed to the **"USB DRIVER"** subdirectory.

(If the software was installed into the default subdirectory, the location would be: **C:\KFLOP4xx\USB DRIVER**)

Click **OK**



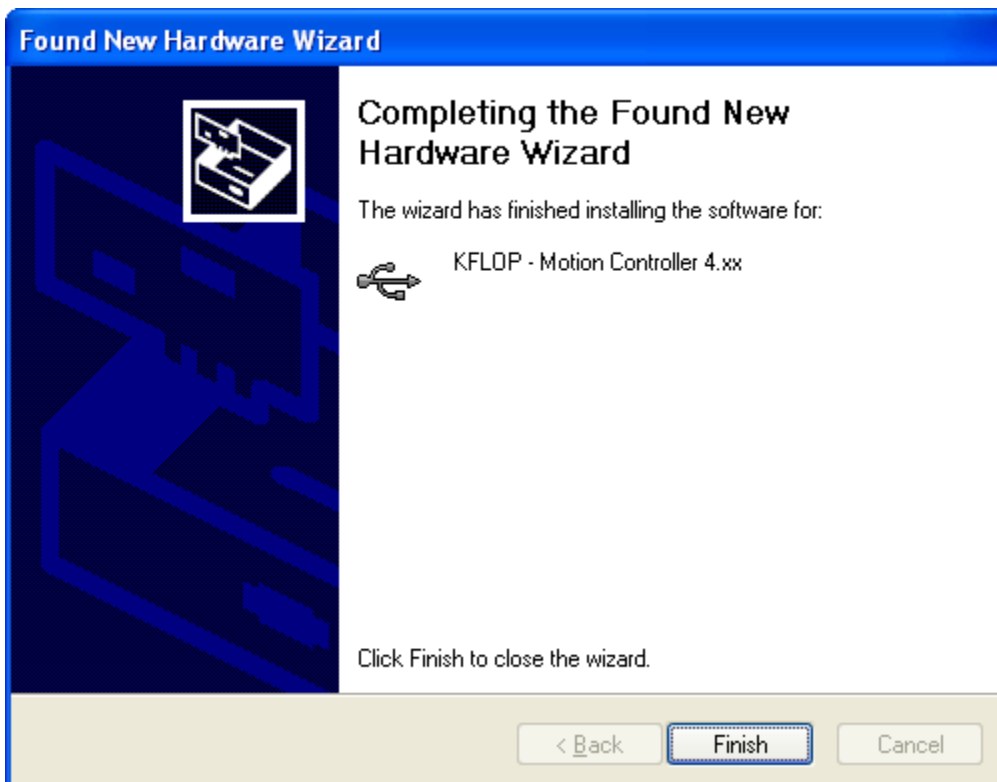
Select **"KFLOP - Motion Controller 4.xx"** and select **Next**



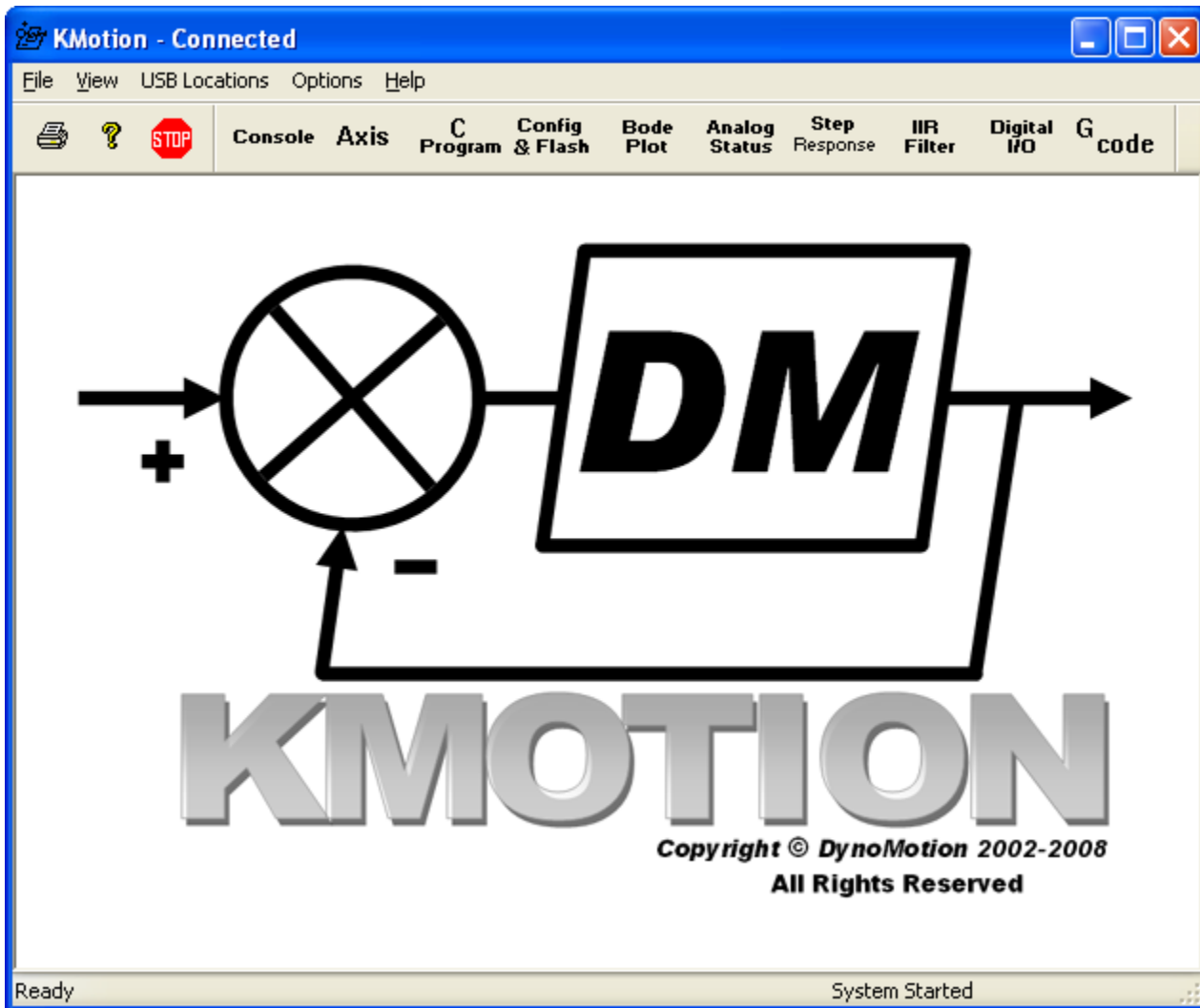
If this screen appears select **Continue Anyway**.



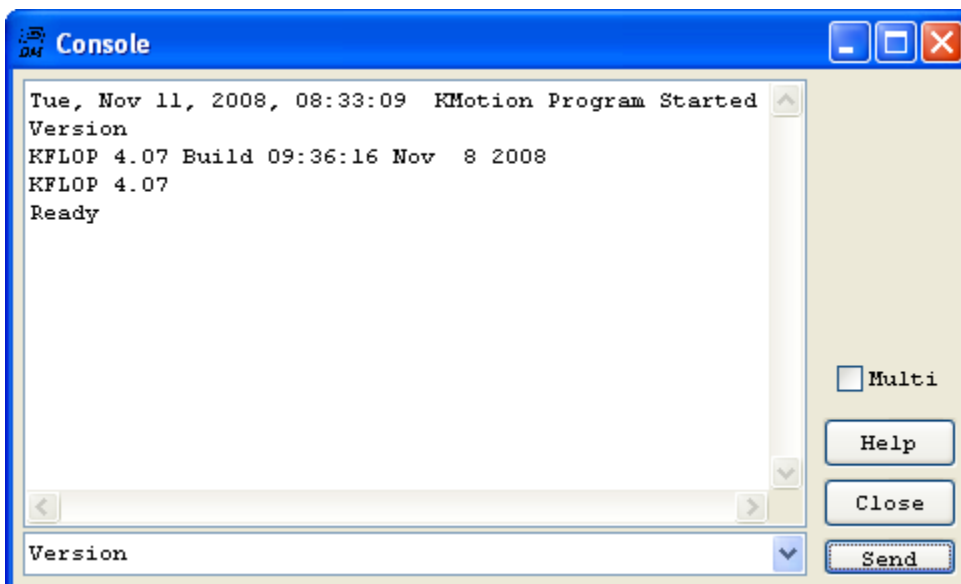
If successful, the above screen should appear. Click **Finish**.



The **KFLOP** Board is now ready for use. To verify proper USB connection to the KFLOP Board, use the Windows™ Start Button to launch the **KMotion** Application.



At the main tool bar select the "Console" button to Display the Console Screen.



Enter the "Version" command and press the "Send" button to send the 'Version" command to the **KFLOP** Board. The following should be displayed indicating successful communication between the PC and **KFLOP** Board.

KFLOP 4.07 Build 09:36:16 Nov 8 2008

KFLOP 4.07

Ready

See the On-Line help for more information on how to connect your motors drives, enter console script commands, compile and run C Programs, execute G Code, and much more!!!

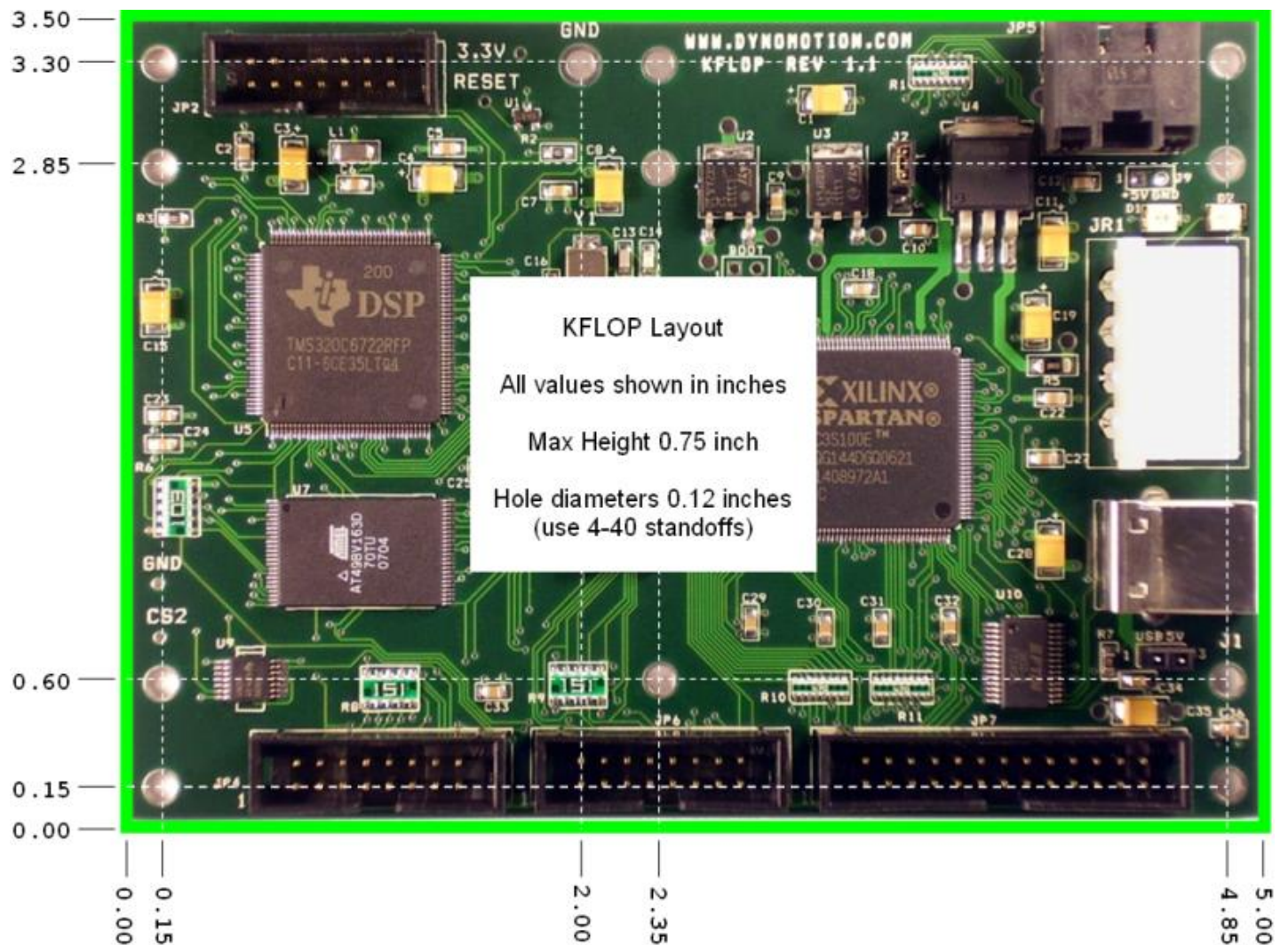
KFLOP Hardware

Function	Parameter	Specification
Processor	CPU Memory	TMS320C67-200MHz DSP 1.2GFLOP 32/64-Bit Native Floating Point FLASH 2 MBytes SDRAM 16 Mbytes
Interface	Host	USB 2.0 Full Speed
Connectors	I/O General Purpose I/O Com Aux#0 IO Aux#1 IO USB System Power	26 pin Header 8 pin RJ45 16 pin Header 16 pin Header Type B Molex 4-pin (Disk drive type)
Servo Loop	Sample Rate Compensation Feed Forward	90 μ s PID + (3) IIR bi-quad Stages/Axis Acceleration + Velocity
Axis	Number Type	8 MicroStep/Servo/Brush/Brushless/StepDirection
Logic Supply	Voltage Max Current Typical Current	+5V \pm 10% 2.5A 0.35 A
User I/O	Digital Encoders	45 Gen Purpose LVTTTL (24 are 5V Tolerant) (4) single-ended, 1 MHz
Environment	Operating Temperature Storage Temperature Humidity	0-40 $^{\circ}$ C 0-40 $^{\circ}$ C 20-90% Relative Humidity, non-condensing
Dimensions	Length Width Height	3.5 inches (89mm) 5.0 inches (127 mm) 0.75inches (19 mm)

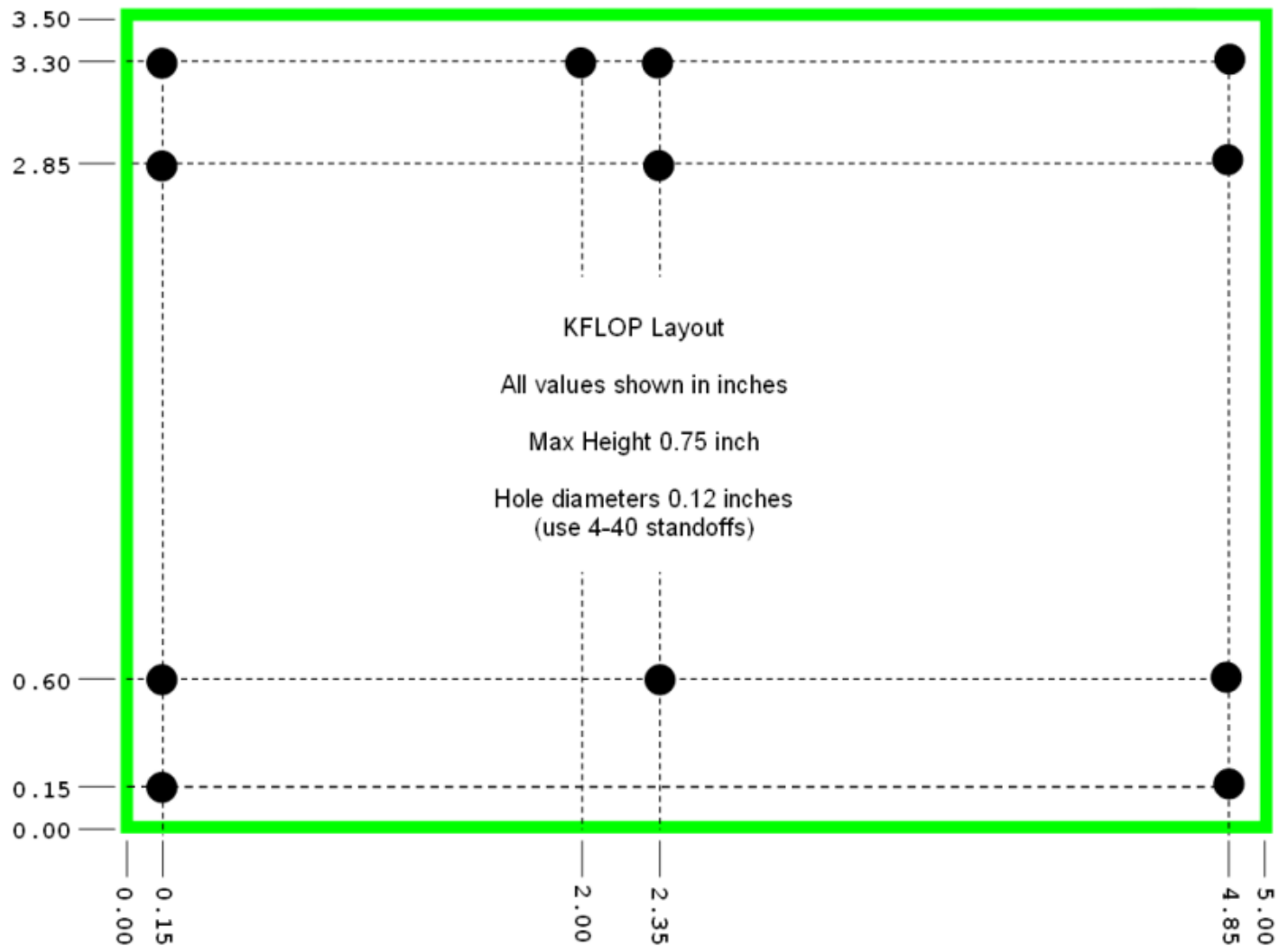
KFLOP Software

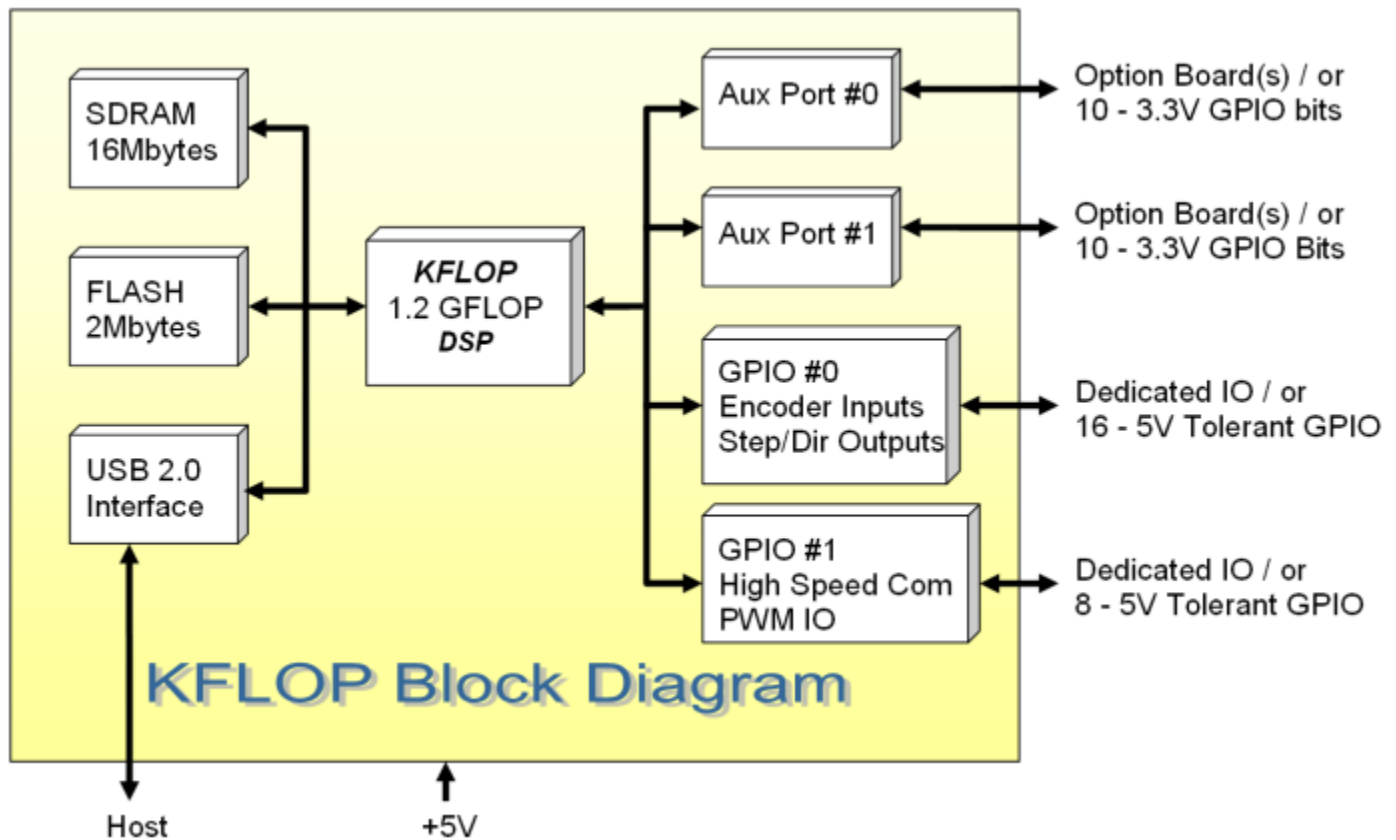
Function	Parameter	Specification
User Programs	Language Number concurrent Stand alone mode	C 7 Yes
Host Requirements	OS Interface	MS Windows™ 2000, MS Windows™ XP USB 2.0
Interface Library	Multi-Thread Multi-Process Multi-Board MS Windows™ VC++ MS Windows™ VB	Yes Yes Yes Supported Supported
C Compiler	TCC67	Included
G Code	Interpreter	Included
Script Language	ASCII Commands	Included
Trajectory Planner	Coordinated Motion	4 Axis
Executive Application	Configuration Tuning User Programs G Code Command Console Status Display	Upload/Download/Save/Load Motor Config Move/Step Response, Bode Plot, Calc Filters Integrated IDE - Edit/Compile/Download/Exec Integrated ASCII Command Entry - Log Console Axis/Analog/Digital

Hide Photo

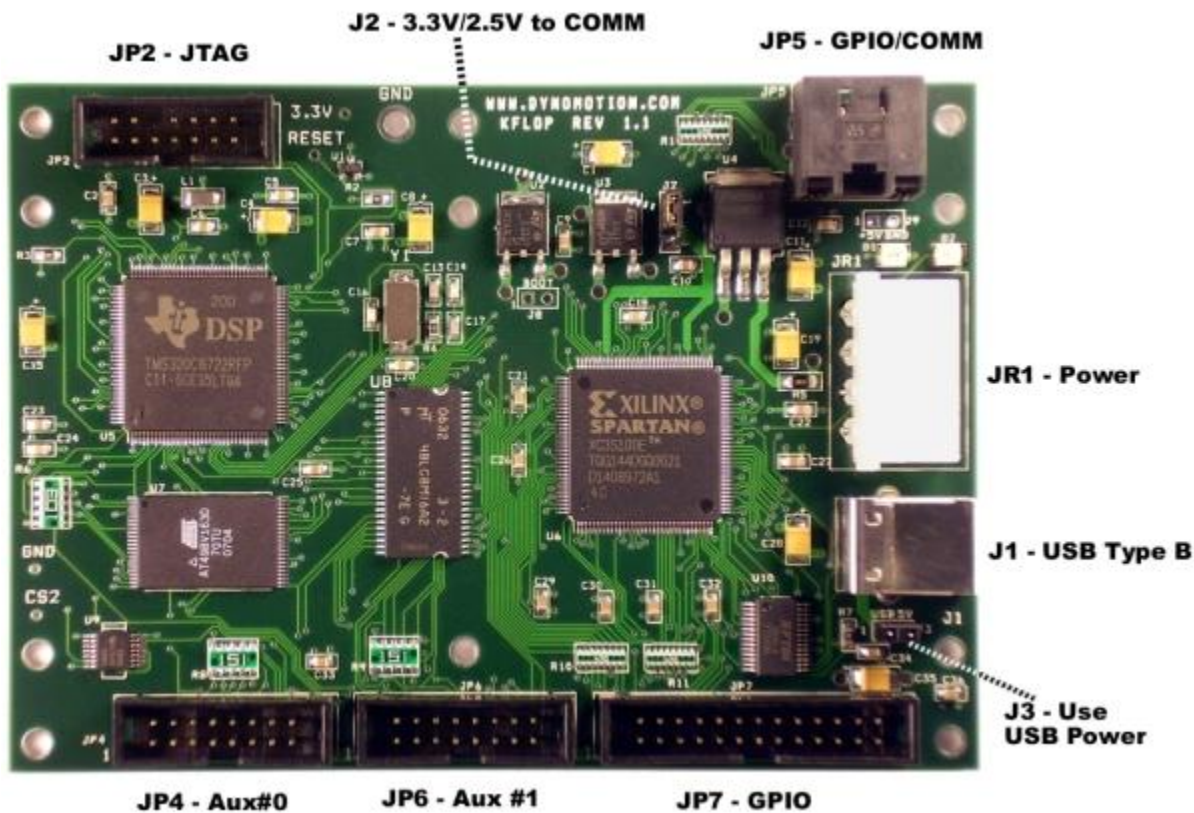


[Show Photo](#)





KFLOP - Connector Pinouts

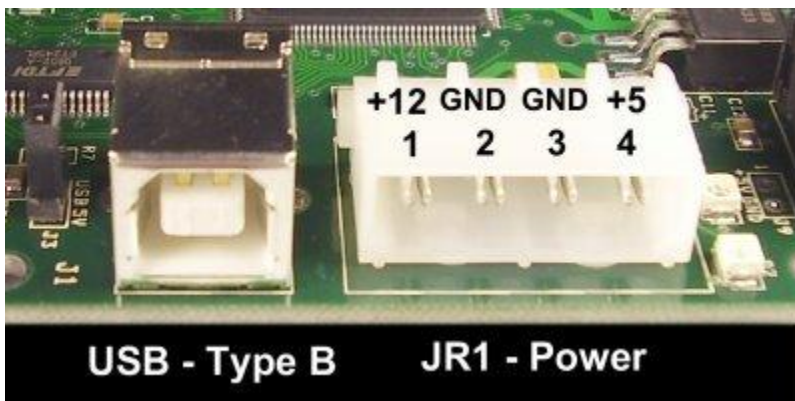
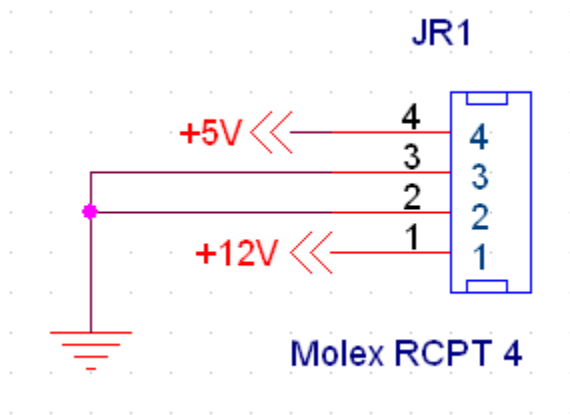


JR1 - +5V Power (regulated +/- 5%)

Typical current = 0.35Amps with no user I/O connected. More current may be required dependent on the amount of Digital I/O and option boards. +5V @ 2.5A should be more than sufficient under all conditions. The +12V input is not used internally by the board, but is routed to pins on the JP4, JP6, and JP7 connectors for the convenience of the user. +5V power is also routed to the same connectors. 5V power may be applied at whichever connector is more convenient. This connector is rated for 6.5Amps per connection.

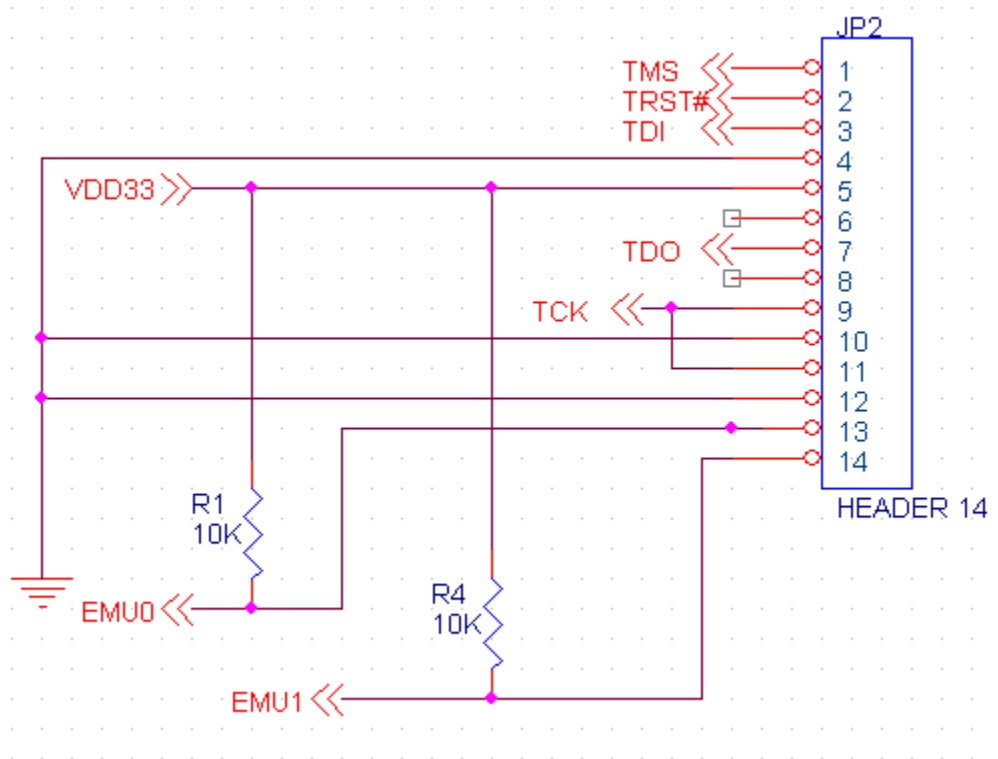
This connector is a standard PC-disk drive power connector which makes it easy to drive the board with an inexpensive PC power supply.

Under some conditions USB power may be used to avoid requiring an external +5V power supply connection. USB specifies a maximum of 500ma may be drawn from the USB cable. KFLOP itself consumes 350ma. If external IO draws less than the remaining 150ma USB supplied power may be used. To utilize USB power connect Jumper J3 and do not supply +5V into any of the KFLOP connectors.



JP2 - JTAG

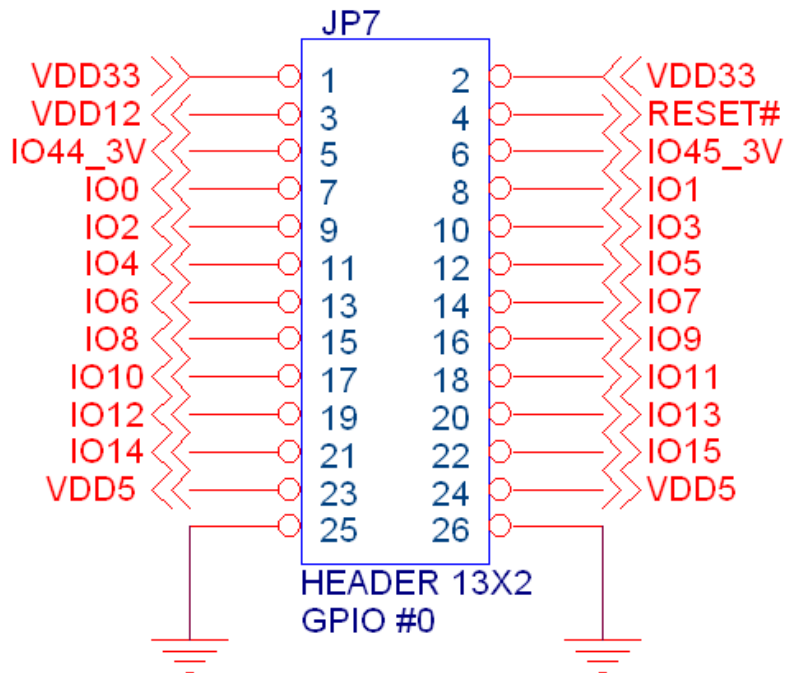
This connector is only used for advanced debugging using an XDS510 JTAG type in circuit emulator. A small amount of regulated 3.3V (<0.5 Amp) is available on this connector if needed for external use.



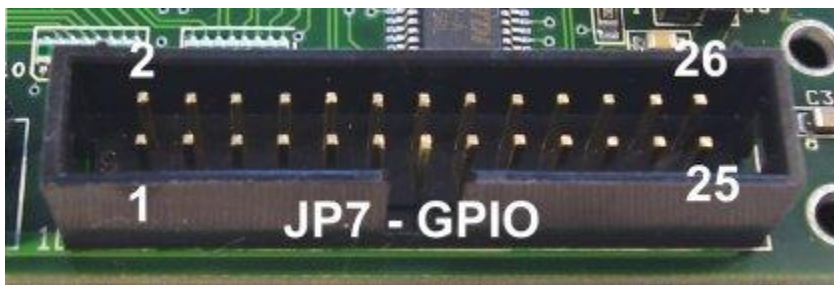
JP7 - Digital IO

18 LVTTTL bi-directional digital I/O, and +5, +15, -15 power supply outputs. Many Digital I/O bits are pre-defined as encoder, home, or limit inputs (see table below) but if not required for the particular application may be used as general purpose I/O. Digital Outputs may sink/source 10 ma. Digital I/O is LVTTTL (3.3V) but is 5 V tolerant.

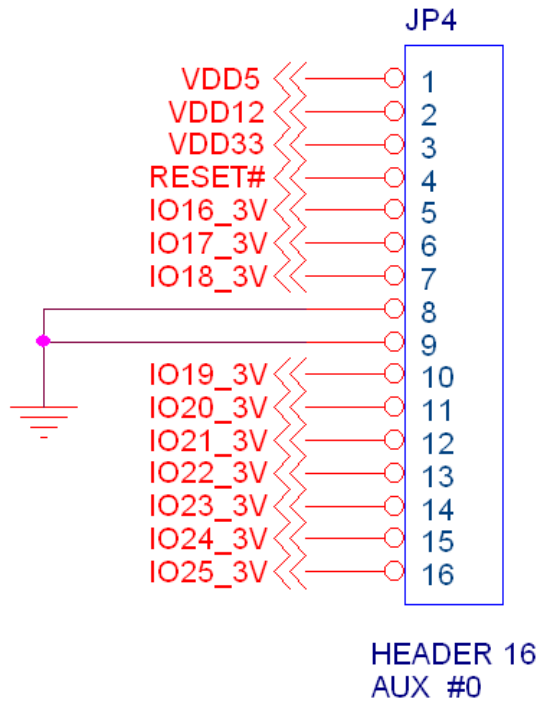
Caution! This connector contains 12V signals. Shorts to low voltage pins will cause permanent damage to the board!



NOTE: DO NOT APPLY
5V to _3V INPUTS

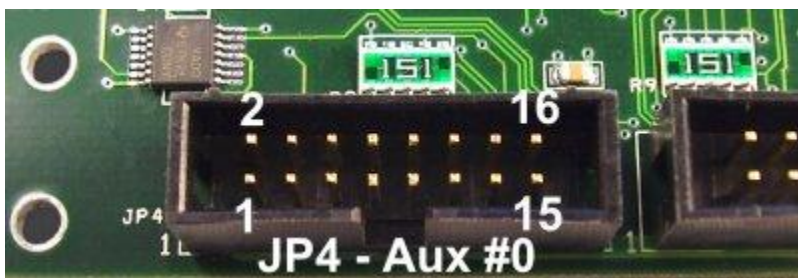


Pin	Name	Description
1	VDD33	+3.3 Volts Output
2	VDD33	+3.3 Volts Output
3	VDD12	+12 Volts Output
4	RESET#	Power up Reset (low true) output
5	IO44	Gen Purpose LVTTL I/O (3.3V Only)
6	IO45	Gen Purpose LVTTL I/O (3.3V Only)
7	IO0	Gen Purpose LVTTL I/O (5V Tolerant) or Axis 0 Encoder Input Phase A
8	IO1	Gen Purpose LVTTL I/O (5V Tolerant) or Axis 0 Encoder Input Phase B
9	IO2	Gen Purpose LVTTL I/O (5V Tolerant) or Axis 1 Encoder Input Phase A
10	IO3	Gen Purpose LVTTL I/O (5V Tolerant) or Axis 1 Encoder Input Phase B
11	IO4	Gen Purpose LVTTL I/O (5V Tolerant) or Axis 2 Encoder Input Phase A
12	IO5	Gen Purpose LVTTL I/O (5V Tolerant) or Axis 2 Encoder Input Phase B
13	IO6	Gen Purpose LVTTL I/O (5V Tolerant) or Axis 3 Encoder Input Phase A
14	IO7	Gen Purpose LVTTL I/O (5V Tolerant) or Axis 3 Encoder Input Phase B
15	IO8	Gen Purpose LVTTL I/O (5V Tolerant) or Axis 0 Home (or Step 0 output)
16	IO9	Gen Purpose LVTTL I/O (5V Tolerant) or Axis 1 Home (or Dir 0 output)
17	IO10	Gen Purpose LVTTL I/O (5V Tolerant) or Axis 2 Home (or Step 1 output)
18	IO11	Gen Purpose LVTTL I/O (5V Tolerant) or Axis 3 Home (or Dir 1 output)
19	IO12	Gen Purpose LVTTL I/O (5V Tolerant) or Axis 0 + Limit (or Step 2 output)
20	IO13	Gen Purpose LVTTL I/O (5V Tolerant) or Axis 0 - Limit (or Dir 2 output)
21	IO14	Gen Purpose LVTTL I/O (5V Tolerant) or Axis 1 + Limit (or Step 3 output)
22	IO15	Gen Purpose LVTTL I/O (5V Tolerant) or Axis 1 - Limit (or Dir 3 output)
23	VDD5	+5 Volts Output
24	VDD5	+5 Volts Output
25	GND	Digital Ground
26	GND	Digital Ground

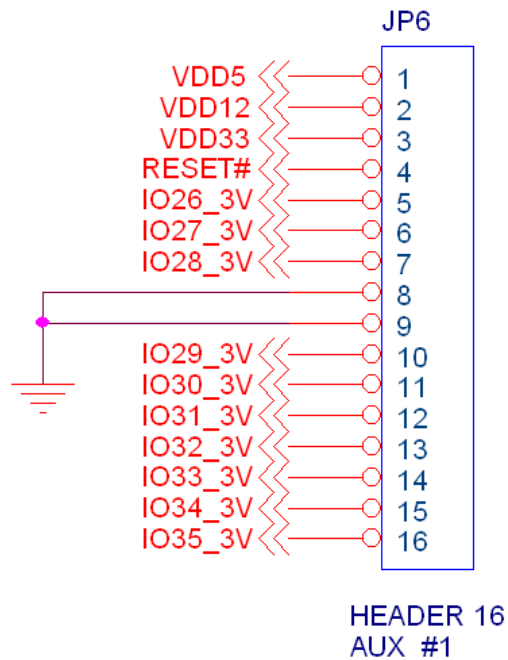
JP4 - Aux Connector #0

NOTE: DO NOT APPLY
5V to _3V INPUTS

Auxiliary connector which supplies power, reset, and 10 digital I/O (LVTTL 3.3V only) which is normally connected to optional expansion daughter boards (ie. SnapAmp 1000). If no expansion module is required, these digital I/O may be used for general purpose use.

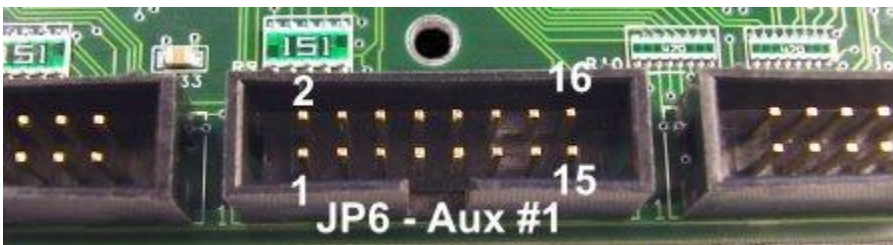


Pin	Name	Description
1	VDD5	+5 Volts Output
2	VDD12	+12 Volts Output
3	VDD33	+3.3 Volts Output
4	RESET#	Power up Reset (low true) output
5	IO16	Gen Purpose LVTTL I/O (3.3V Only)
6	IO17	Gen Purpose LVTTL I/O (3.3V Only)
7	IO18	Gen Purpose LVTTL I/O (3.3V Only)
8	GND	Digital Ground
9	GND	Digital Ground
10	IO19	Gen Purpose LVTTL I/O (3.3V Only)
11	IO20	Gen Purpose LVTTL I/O (3.3V Only)
12	IO21	Gen Purpose LVTTL I/O (3.3V Only)
13	IO22	Gen Purpose LVTTL I/O (3.3V Only)
14	IO23	Gen Purpose LVTTL I/O (3.3V Only)
15	IO24	Gen Purpose LVTTL I/O (3.3V Only)
16	IO25	Gen Purpose LVTTL I/O (3.3V Only)

JP6 - Aux Connector #1

NOTE: DO NOT APPLY
5V to _3V INPUTS

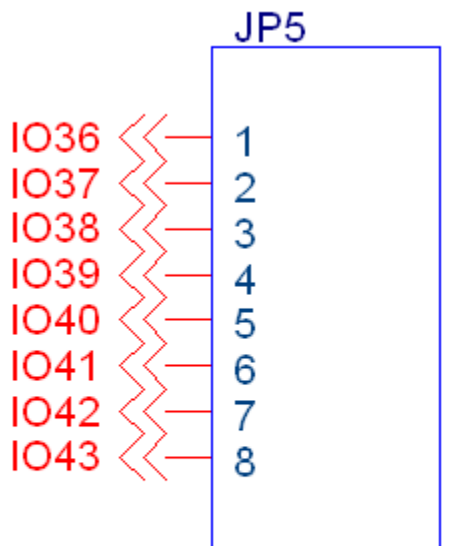
Auxiliary connector which supplies power, reset, and 10 digital I/O (LVTTTL 3.3V only) which is normally connected to optional expansion daughter boards (ie. SnapAmp 1000). If no expansion module is required, these digital I/O may be used for general purpose use.



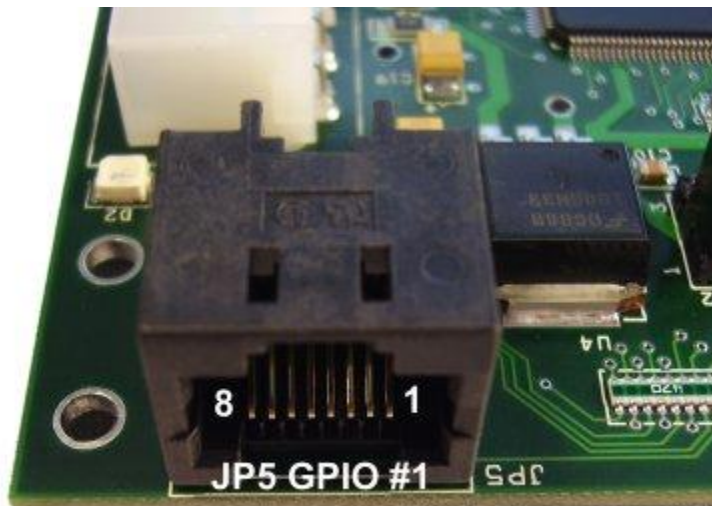
Pin	Name	Description
1	VDD5	+5 Volts Output
2	VDD12	+12 Volts Output
3	VDD33	+3.3 Volts Output
4	RESET#	Power up Reset (low true) output
5	IO26	Gen Purpose LVTTL I/O (3.3V Only) or PWM0 out
6	IO27	Gen Purpose LVTTL I/O (3.3V Only) or PWM1 out
7	IO28	Gen Purpose LVTTL I/O (3.3V Only) or PWM2 out
8	GND	Digital Ground
9	GND	Digital Ground
10	IO29	Gen Purpose LVTTL I/O (3.3V Only) or PWM3 out
11	IO30	Gen Purpose LVTTL I/O (3.3V Only) or PWM4 out
12	IO31	Gen Purpose LVTTL I/O (3.3V Only) or PWM5 out
13	IO32	Gen Purpose LVTTL I/O (3.3V Only) or PWM6 out
14	IO33	Gen Purpose LVTTL I/O (3.3V Only) or PWM7 out
15	IO34	Gen Purpose LVTTL I/O (3.3V Only)
16	IO35	Gen Purpose LVTTL I/O (3.3V Only)

JP5 - GPIO #1 / LV Differential Connector

Low Voltage Differential RJ45 Connector. This connector is intended for high-speed low-voltage differential communication over a twisted pair cable use. However it may also be used as 8 General Purpose digital I/O (LVTTTL 5V Tolerant)



RJHSE-5080
GPIO #1
5V TOLERANT



Pin	Name	Description
1	IO36	Gen Purpose LVTTL I/O (5V Tolerant) or (or Step 4 output) or (Axis 4 Encoder Input Phase A)
2	IO37	Gen Purpose LVTTL I/O (5V Tolerant) or (or Dir 4 output) or (Axis 4 Encoder Input Phase B)
3	IO38	Gen Purpose LVTTL I/O (5V Tolerant) or (or Step 5 output) or (Axis 5 Encoder Input Phase A)
4	IO39	Gen Purpose LVTTL I/O (5V Tolerant) or (or Dir 5 output) or (Axis 5 Encoder Input Phase B)
5	IO40	Gen Purpose LVTTL I/O (5V Tolerant) or (or Step 6 output) or (Axis 6 Encoder Input Phase A)
6	IO41	Gen Purpose LVTTL I/O (5V Tolerant) or (or Dir 6 output) or (Axis 6 Encoder Input Phase B)
7	IO42	Gen Purpose LVTTL I/O (5V Tolerant) or (or Step 7 output) or (Axis 7 Encoder Input Phase A)
8	IO43	Gen Purpose LVTTL I/O (5V Tolerant) or (or Dir 7 output) or (Axis 7 Encoder Input Phase B)

Analog I/O Status Screen

Analog I/O

Snap0 Snap1 Kanalog

ADCs		Supplies		PWMs	
#8	8171 = -0.090 amps	#4	35839 = 55.7 V	OR=	0.0%
#9	8181 = -0.047 amps	#5	35773 = 55.6 V	OR=	0.0%
#10	8195 = 0.013 amps	Temperature		OR=	0.0%
#11	8202 = 0.043 amps	#4	180 = 22.5 C	OR=	0.0%
		#5	183 = 22.9 C		

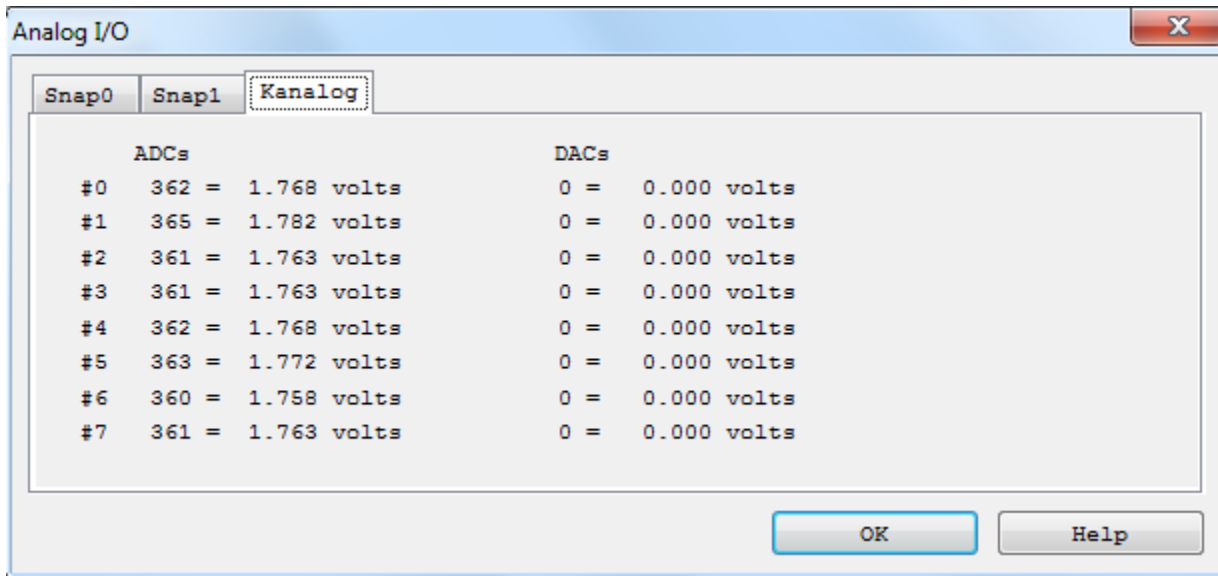
OK Help

Analog I/O

Snap0 Snap1 Kanalog

ADCs		Supplies		PWMs	
#12	8171 = -0.090 amps	#6	35839 = 55.7 V	OR=	0.0%
#13	8181 = -0.047 amps	#7	35773 = 55.6 V	OR=	0.0%
#14	8195 = 0.013 amps	Temperature		OR=	0.0%
#15	8202 = 0.043 amps	#6	180 = 22.5 C	OR=	0.0%
		#7	183 = 22.9 C		

OK Help



The **Analog I/O Status Screen** displays various analog measurements and commands including:

- Kanalog DAC Settings
- Kanalog ADC Readings
- SnapAmp Measured current flow per motor coil
- SnapAmp Power Supply Voltages
- SnapAmp Mosfet Temperatures
- SnapAmp PWM power amp settings

KFLOP itself has no Analog capability and when operated alone with no option cards the Analog I/O Screen will not display. If one or two **SnapAmp** option boards are installed and checked on the option menu, tabs will allow display **of SnapAmp** Status. Current measurement for each full bridge (motor coil), Power supply voltage for each side, Temperature of each side, and PWM setting for each PWM. If Kanalog is installed and checked on the option menu a tab will allow displaying Kanalog Analog Status.

PWM's

The state of each **SnapAmp's** four PWM (Pulse Width Modulators) are displayed in the top right area of the status screen. The PWM's are connected to *Full Bridge Drivers* to allow direct control of various motors or loads. See the description of [KMotion's Power Amplifiers and PWM's](#) for details. The PWM's may operate independently to drive a full bridge driver, or they may function as a pair of PWM's connected to a pair of Full Bridge drivers to drive a 4 phase stepper motor or a [3 phase load](#).

PWMs may be assigned to an axis by changing the OutputChan0 and OutputChan1 parameters for an axis. Only consecutive even and odd PWMS may be paired to drive a 3 or 4 phase phase load.

There are several possible modes for each PWM channels:

- Normal Independent
- Recirculating Independent
- 3 Phase - paired
- Current Feedback

If a PWM channel is operating in Normal mode, the PWM channel status will show the value in counts (-255 ... +255) and the percent of full scale.

If a PWM channel is operating in Recirculating mode, the PWM channel status will show the value in counts (-511 ... +511), followed by an "R", and the percent of full scale.

A PWM channel may also operate in current loop mode. In current loop mode the PWM duty cycle is automatically adjusted in order to maintain the commanded current. The PWM channel status will show the command value in counts (-4095 ... +4095), followed by a "C", and the percent of full scale.

If a pair of PWM channels is operating in 3 Phase mode, the PWM channel status will show the value in counts (-230 ... 230) after the first PWM channel and the phase angle in degrees after second PWM channel.

The example status below shows PWM channels 8 and 9 operating in 3 phase mode, PWM channel 10 operating in Normal mode, and PWM channels 11 operating in Recirculating mode.

```
PWMs
100 = 43.5%
120 degrees
50 = 19.5%
200R= 39.1%
```

Digital I/O Screen

Digital I/O

KFLOP SnapAmp0 SnapAmp1 Kanalog

Output	Bit	State	Output	Bit	State
<input type="checkbox"/>	0 Enc 0 A	<input checked="" type="checkbox"/>	<input type="checkbox"/>	26 Aux1	<input type="checkbox"/>
<input type="checkbox"/>	1 Enc 0 B	<input checked="" type="checkbox"/>	<input type="checkbox"/>	27 Aux1	<input type="checkbox"/>
<input type="checkbox"/>	2 Enc 1 A	<input checked="" type="checkbox"/>	<input type="checkbox"/>	28 Aux1	<input type="checkbox"/>
<input type="checkbox"/>	3 Enc 1 B	<input type="checkbox"/>	<input type="checkbox"/>	29 Aux1	<input type="checkbox"/>
<input type="checkbox"/>	4 Enc 2 A	<input type="checkbox"/>	<input type="checkbox"/>	30 Aux1	<input type="checkbox"/>
<input type="checkbox"/>	5 Enc 2 B	<input type="checkbox"/>	<input type="checkbox"/>	31 Aux1	<input type="checkbox"/>
<input type="checkbox"/>	6 Enc 3 A	<input checked="" type="checkbox"/>	<input type="checkbox"/>	32 Aux1	<input type="checkbox"/>
<input type="checkbox"/>	7 Enc 3 B	<input checked="" type="checkbox"/>	<input type="checkbox"/>	33 Aux1	<input type="checkbox"/>
<input checked="" type="checkbox"/>	8 Home 0	<input type="checkbox"/>	<input type="checkbox"/>	34 Aux1	<input type="checkbox"/>
<input checked="" type="checkbox"/>	9 Home 1	<input type="checkbox"/>	<input type="checkbox"/>	35 Aux1	<input type="checkbox"/>
<input checked="" type="checkbox"/>	10 Home 2	<input type="checkbox"/>			
<input checked="" type="checkbox"/>	11 Home 3	<input checked="" type="checkbox"/>	<input type="checkbox"/>	36 Com	<input type="checkbox"/>
<input checked="" type="checkbox"/>	12 LIM + 0	<input type="checkbox"/>	<input type="checkbox"/>	37 Com	<input type="checkbox"/>
<input checked="" type="checkbox"/>	13 LIM - 0	<input type="checkbox"/>	<input type="checkbox"/>	38 Com	<input type="checkbox"/>
<input type="checkbox"/>	14 LIM + 1	<input type="checkbox"/>	<input type="checkbox"/>	39 Com	<input type="checkbox"/>
<input checked="" type="checkbox"/>	15 LIM - 1	<input type="checkbox"/>	<input type="checkbox"/>	40 Com	<input type="checkbox"/>
			<input type="checkbox"/>	41 Com	<input type="checkbox"/>
<input checked="" type="checkbox"/>	16 Aux0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	42 Com	<input type="checkbox"/>
<input checked="" type="checkbox"/>	17 Aux0	<input type="checkbox"/>	<input type="checkbox"/>	43 Com	<input type="checkbox"/>
<input checked="" type="checkbox"/>	18 Aux0	<input type="checkbox"/>			
<input checked="" type="checkbox"/>	19 Aux0	<input type="checkbox"/>	<input type="checkbox"/>	44	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	20 Aux0	<input type="checkbox"/>	<input type="checkbox"/>	45	<input type="checkbox"/>
<input type="checkbox"/>	21 Aux0	<input type="checkbox"/>			
<input type="checkbox"/>	22 Aux0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	46 LED0	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	23 Aux0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	47 LED1	<input checked="" type="checkbox"/>
<input type="checkbox"/>	24 Aux0	<input type="checkbox"/>			
<input type="checkbox"/>	25 Aux0	<input type="checkbox"/>			

OK Help

Digital I/O

KFLOP SnapAmp0 SnapAmp1 Kanalog

Bit		State	Output	Bit		State
64	Diff 0 Enc 0 A	<input checked="" type="checkbox"/>	<input type="checkbox"/>	80	GPIO 0	<input type="checkbox"/>
65	Diff 1 Enc 0 B	<input checked="" type="checkbox"/>	<input type="checkbox"/>	81	GPIO 1	<input type="checkbox"/>
66	Diff 2 Enc 1 A	<input checked="" type="checkbox"/>	<input type="checkbox"/>	82	GPIO 2	<input type="checkbox"/>
67	Diff 3 Enc 1 B	<input checked="" type="checkbox"/>	<input type="checkbox"/>	83	GPIO 3	<input type="checkbox"/>
68	Diff 4 Enc 2 A	<input checked="" type="checkbox"/>	<input type="checkbox"/>	84	GPIO 4	<input type="checkbox"/>
69	Diff 5 Enc 2 B	<input checked="" type="checkbox"/>	<input type="checkbox"/>	85	GPIO 5	<input type="checkbox"/>
70	Diff 6 Enc 3 A	<input checked="" type="checkbox"/>	<input type="checkbox"/>	86	GPIO 6	<input type="checkbox"/>
71	Diff 7 Enc 3 B	<input checked="" type="checkbox"/>	<input type="checkbox"/>	87	GPIO 7	<input type="checkbox"/>
72	Opto 0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	88	GPIO 8	<input type="checkbox"/>
73	Opto 1	<input checked="" type="checkbox"/>	<input type="checkbox"/>	89	GPIO 9	<input type="checkbox"/>
74	Opto 2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	90	GPIO 10	<input type="checkbox"/>
75	Opto 3	<input checked="" type="checkbox"/>	<input type="checkbox"/>	91	GPIO 11	<input type="checkbox"/>
76	Opto 4	<input checked="" type="checkbox"/>	<input type="checkbox"/>	92	GPIO 12	<input type="checkbox"/>
77	Opto 5	<input checked="" type="checkbox"/>	<input type="checkbox"/>	93	GPIO 13	<input type="checkbox"/>
78	Opto 6	<input checked="" type="checkbox"/>				
79	Opto 7	<input checked="" type="checkbox"/>				

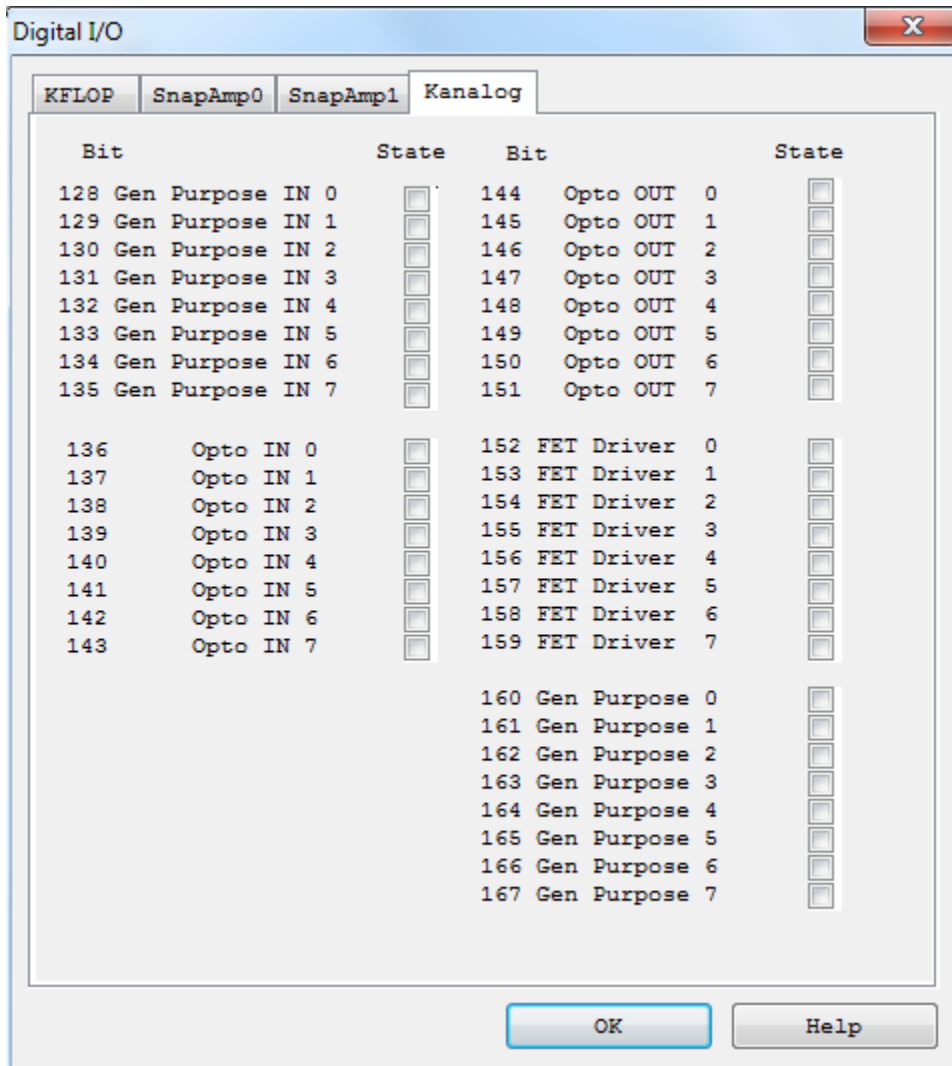
OK Help

Digital I/O

KFLOP SnapAmp0 SnapAmp1 Kanalog

Bit		State	Output	Bit		State
96	Diff 0 Enc 0 A	<input type="checkbox"/>	<input type="checkbox"/>	112	GPIO 0	<input type="checkbox"/>
97	Diff 1 Enc 0 B	<input type="checkbox"/>	<input type="checkbox"/>	113	GPIO 1	<input type="checkbox"/>
98	Diff 2 Enc 1 A	<input type="checkbox"/>	<input type="checkbox"/>	114	GPIO 2	<input type="checkbox"/>
99	Diff 3 Enc 1 B	<input type="checkbox"/>	<input type="checkbox"/>	115	GPIO 3	<input type="checkbox"/>
100	Diff 4 Enc 2 A	<input type="checkbox"/>	<input type="checkbox"/>	116	GPIO 4	<input type="checkbox"/>
101	Diff 5 Enc 2 B	<input type="checkbox"/>	<input type="checkbox"/>	117	GPIO 5	<input type="checkbox"/>
102	Diff 6 Enc 3 A	<input type="checkbox"/>	<input type="checkbox"/>	118	GPIO 6	<input type="checkbox"/>
103	Diff 7 Enc 3 B	<input type="checkbox"/>	<input type="checkbox"/>	119	GPIO 7	<input type="checkbox"/>
104	Opto 0	<input type="checkbox"/>	<input type="checkbox"/>	120	GPIO 8	<input type="checkbox"/>
105	Opto 1	<input type="checkbox"/>	<input type="checkbox"/>	121	GPIO 9	<input type="checkbox"/>
106	Opto 2	<input type="checkbox"/>	<input type="checkbox"/>	122	GPIO 10	<input type="checkbox"/>
107	Opto 3	<input type="checkbox"/>	<input type="checkbox"/>	123	GPIO 11	<input type="checkbox"/>
108	Opto 4	<input type="checkbox"/>	<input type="checkbox"/>	124	GPIO 12	<input type="checkbox"/>
109	Opto 5	<input type="checkbox"/>	<input type="checkbox"/>	125	GPIO 13	<input type="checkbox"/>
110	Opto 6	<input type="checkbox"/>				
111	Opto 7	<input type="checkbox"/>				

OK Help



The **Digital I/O Screen** displays and allows changes to the current state of **KFLOP** digital I/O bits. Digital IO bits are present on the **KFLOP** as well as any additional option cards (such as with two SnapAmps or Kanalog as shown above). IO bits are numbered 0-167.

KMotion has a number of digital I/O bits that may be used as *GPIO* (General Purpose Inputs or Outputs) or as specific dedicated functions (ie. encoder inputs). There are 46 bits that may be utilized as GPIO (bits 0 - 45). Each bit may be independently defined as either an input or an output. On Power UP **KMotion** defines all I/O as inputs by default. Any bit may be configured as an output by checking the corresponding box in the "Output" columns. Alternately, the bits may be configured by a C program running within the **KMotion** (See [SetBitDirection\(\)](#)) or by Script commands (See [SetBitDirection](#)) sent to the **KMotion**.

KFLOP bits 0-15 are connected to the JP7 connector normally which may be used to connect to auxiliary option boards such as Kanalog. These bits are available for User use if no option cards are requiring them in the system. **If they are used to communicate to an option board care should be taken to not issue any User IO commands to these bits.**

KFLOP bits 16 - 25 are connected to the Aux#0 connector normally used as a high-speed communication bus to connect to auxiliary option boards such as SnapAmps. These bits are available for User use if no option cards are requiring them in the system. **If they are used to communicate to an option board care should be taken to not issue any User IO commands to these bits.**

The *State* of each I/O bit may be observed in the corresponding checkbox under the "State" columns. If the bit is defined as an output, clicking on the "State" checkbox will toggle the bit. Alternately, the bits may be read, set, or cleared by a C program running within the **KMotion** (See [ReadBit\(\)](#), [SetBit\(\)](#), [ClearBit\(\)](#), or [SetStateBit\(\)](#)) or by Script commands (See [ReadBit](#), [SetBit](#), [ClearBit](#), or [SetStateBit](#)) sent to the **KMotion**.

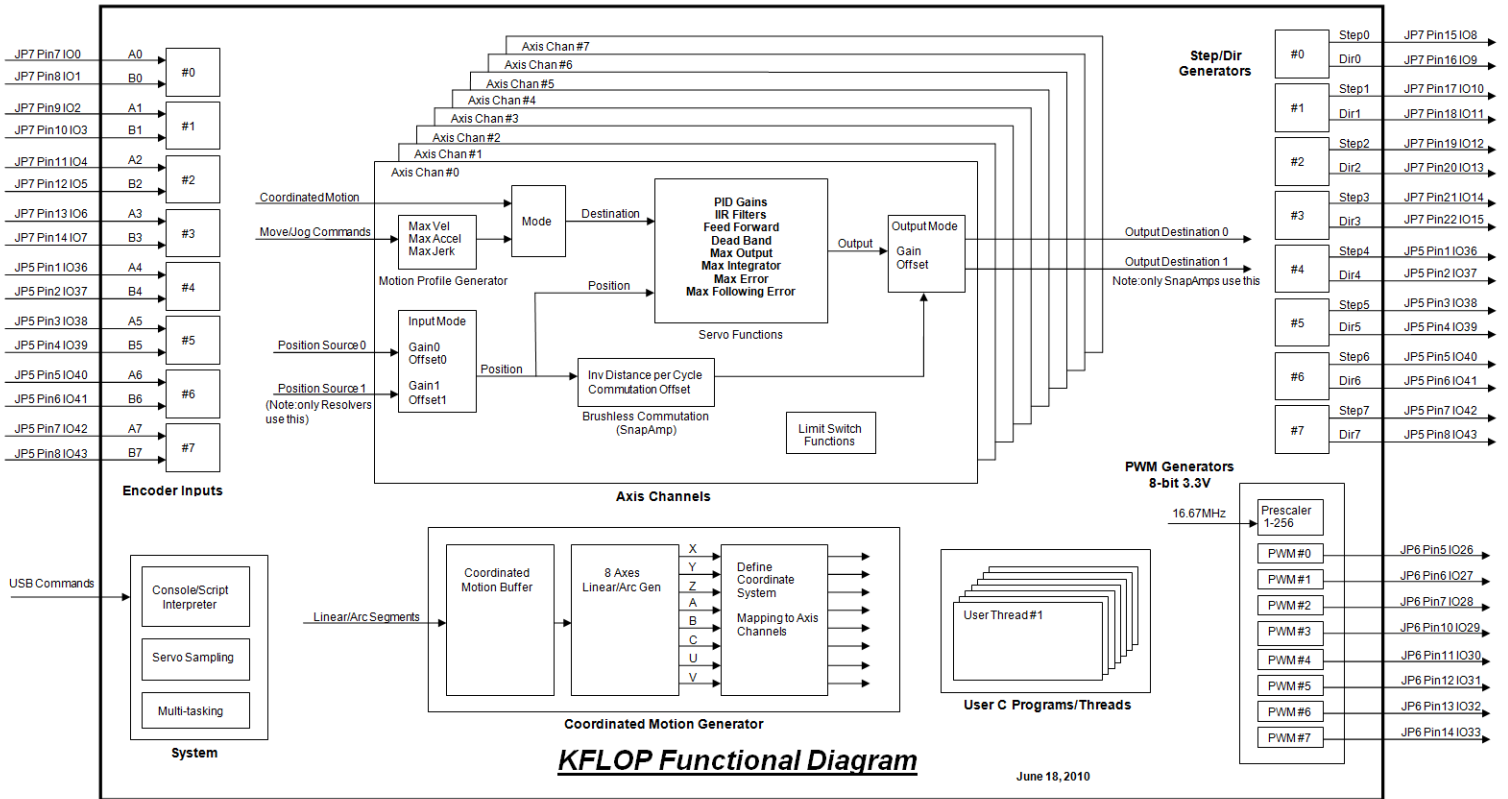
Additionally, *buffered* commands may change the state of Digital I/O bits. *Buffered* I/O commands are I/O commands that are inserted into the coordinated motion buffer. When it is required that I/O bits be changed at exact times within a motion sequence, *buffered* I/O commands may be inserted into the motion buffer (see [SetBitBuf](#), [ClearBitBuf](#), and [SetStateBitBuf](#)). In this case the I/O commands occur when they are encountered within the motion sequence. The **KMotion** GCode interpreter allows buffered I/O commands to be inserted within motion sequences by using a special form of GCode comment (See [buffered GCode Commands](#)).

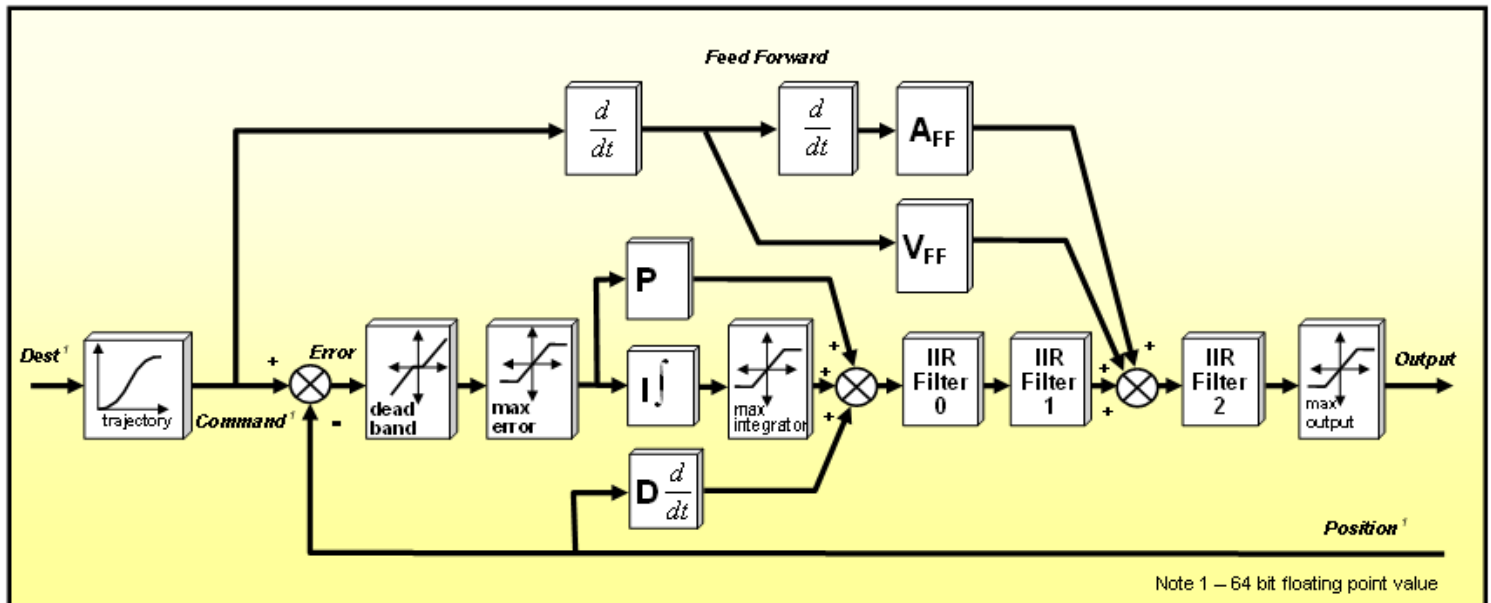
See the [KFLOP Hardware Connector Descriptions](#) , [SnapAmp Hardware Connector Descriptions](#) , or [Kanalogue HardwareConnector Descriptions](#) for which IO bits are connected to the various connectors.

Caution: Shorting High Voltage (greater than 3.3V or 5V depending on pin tolerance) to any Digital I/O bit will be likely to cause permanent board damage.

Digital I/O bits 46 and 47 are dedicated to the control of the two LED's on **KFLOP**. These two outputs are configured as outputs and tuned on when **KFLOP** powers up. They are available for use as User status if desired.

KFLOP Functional Diagram





KMotion 2.2 Servo Flow Diagram

KMotion Quick Reference

KMotionDLL

Send Commands

[WriteLine](#)

[WriteLineReadLine](#)

[ReadLineTimeout](#)

Board Locks

[WaitToken](#)

[KMotionLock](#)

[ReleaseToken](#)

[Failed](#)

Console

[ServiceConsole](#)

[SetConsoleCallback](#)

Coff Loader

[LoadCoff](#)

Compiler

[CompileAndLoadCoff](#)

USB

[ListLocations](#)

```
int WriteLine(int board, const char *s);
```

Writes a null terminated string of characters to a specified **KMotion** Board. There is no wait for any response.

Return Value

0 if successful, non-zero if unsuccessful (invalid board specified)

Parameters

board

specifies which board in the system the command applies to

s

Null terminated string to send

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM;
if (KM.WriteLine(0, "Move0=1000") MyError());
```

```
int WriteLineReadLine(int board, const char *s, char
*response);
```

Writes a null terminated string of characters to a specified **KMotion** Board. Waits for a response string. This command is thread safe. It waits for the token for the specified board, sends the command, waits for the response, then releases the board.

Return Value

0 if successful, non-zero if unsuccessful (invalid board specified, timeout on the response)

Parameters

board

specifies which board in the system the command applies to

s

Null terminated string to send

response

Buffer to receive the null terminated string received as response

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM;
char resp[256];
while
{
    if (KM.WriteLineReadLine(0, "CheckDone0", resp)
    MyError();
    if (strcmp(resp, "1")==0) break;
}
```

```
int ReadLineTimeout(int board, char *buf, int
TimeOutms);
```

Waits for a response string from a previously issued command. Note in a multi-process or multi thread environment the **KMotion** board should be locked prior to issuing a command that has a response(s), Otherwise there is a possibility that another process or thread may receive the expected response.

Return Value

0 if successful, non-zero if unsuccessful (invalid board specified, timeout on the response)

Parameters

board

specifies which board in the system the command applies to

buf

Buffer to receive the Null terminated string received as response

TimeOutms

Amount of time to receive a response

Example

```
#include "KMotionDLL.h"

CKMotionDLL KM;

char resp1[256];

char resp2[256];

char resp3[256];

// first get the token for the board to allow
uninterrupted access
if (KM.WaitToken(0) != KMOTION_LOCKED) MyError();

// tell the board to send 24 (32 bit) words at offset 0
if (KM.WriteLine(0, "GetGatherHex 0 24")) MyError();

// receive the data (8 hex words per line)
if (KM.ReadLineTimeout(0, resp1)) MyError();
if (KM.ReadLineTimeout(0, resp2)) MyError();
if (KM.ReadLineTimeout(0, resp3)) MyError();

// release our access to the board

KM.ReleaseToken(board);
```

int WaitToken(int board);

Waits until the token for the specified **KMotion** board can be obtained. Call this function whenever uninterrupted access to a **KMotion** board is required. For example before a command where several lines of response will be returned. Release the token as quickly as possible by calling the [ReleaseToken](#) function as all other access to the locked board will be blocked until released.

Return Value

0 if successful, non-zero if unsuccessful (invalid board specified)

Parameters

board

specifies which board in the system the command applies to

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM;
char resp1[256];
char resp2[256];
char resp3[256];
// first get the token for the board to allow
uninterrupted access
if (KM.WaitToken(0) != KMOTION_LOCKED) MyError();

// tell the board to send 24 (32 bit ) words at offset
0
if (KM.WriteLine(0, "GetGatherHex 0 24")) MyError();

// receive the data (8 hex words per line)
if (KM.ReadLineTimeout(0, resp1)) MyError();
if (KM.ReadLineTimeout(0, resp2)) MyError();
if (KM.ReadLineTimeout(0, resp3)) MyError();

// release our access to the board

KM.ReleaseToken(board);
```

int KMotionLock(int board);

Attempts to obtain the token of the specified **KMotion** board.. Call this function whenever uninterrupted access to a **KMotion** board is required. For example before a command where several lines of response will be returned. Release the token as quickly as possible by calling the [ReleaseToken](#) function as all other access to the locked board will be blocked until released. This function is similar to the [WaitToken](#) function, except that it returns immediately (instead of waiting) if the board is already locked.

Return Value

KMOTION_LOCKED=0, // (and token is locked) if KMotion is available for use
KMOTION_IN_USE=1, // if already in use
KMOTION_NOT_CONNECTED=2 // if error or not able to connect

Parameters

board

specifies which board in the system the command applies to

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM;
char resp1[256];
char resp2[256];
char resp3[256];
int result;

// first get the token for the board to allow
uninterrupted access
do
{
    result = KM.KMotionLock(0);
    if (result == KMOTION_NOT_CONNECTED) MyError();
    if (result == KMOTION_IN_USE)
        DoOtherProcessing();
}

// tell the board to send 24 (32 bit) words at offset 0
if (KM.WriteLine(0,"GetGatherHex 0 24")) MyError();

// receive the data (8 hex words per line)
if (KM.ReadLineTimeout(0,resp1)) MyError();
if (KM.ReadLineTimeout(0,resp2)) MyError();
if (KM.ReadLineTimeout(0,resp3)) MyError();

// release our access to the board

KM.ReleaseToken(board);
```

void ReleaseToken(int board);

Releases the previously obtained token of the specified **KMotion** board. See [WaitToken](#) and [LockKMotion](#) functions. The token should always be released as quickly as possible as all other access to the locked board will be blocked until released.

Return Value

none - the function can not fail

Parameters

board

specifies which board in the system the command applies to

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM;
char resp1[256];
char resp2[256];
char resp3[256];
int result;

// first get the token for the board to allow
uninterrupted access
do
{
    result = KM.KMotionLock(0);
    if (result == KMOTION_NOT_CONNECTED) MyError();
    if (result == KMOTION_IN_USE)
DoOtherProcessing();
}

// tell the board to send 24 (32 bit) words at offset 0
if (KM.WriteLine(0,"GetGatherHex 0 24")) MyError();

// receive the data (8 hex words per line)
if (KM.ReadLineTimeout(0,resp1)) MyError();
if (KM.ReadLineTimeout(0,resp2)) MyError();
if (KM.ReadLineTimeout(0,resp3)) MyError();

// release our access to the board

KM.ReleaseToken(board);
```

```
int Failed(int board);
```

This function should be called whenever an error is detected with a **KMotion** board. This function disconnects the driver, flags the board as disconnected, and displays the error message shown below. A user program may typically detect a timeout error or invalid data error if the **KMotion** board is powered down or unplugged while communication is in progress. Calling this function will force any subsequent attempts to access the board to wait for a board to be connected, re-connect, flush any buffers, etc...

"Read Failed - Auto Disconnect"

Return Value

always 0 - the function can not fail

Parameters

board

specifies which board in the system the command applies to

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM;

if (KM.KMotionLock(0) == KMOTION_LOCKED) // see if we
can get access
{
    // upload bulk status

    if (UploadStatus())
    {
        // error reading status
        KM.Failed(0);
    }
}
```

```
    }  
    KM.ReleaseToken(0);  
}
```

```
int LoadCoff(int board, const char *Name, unsigned int  
*EntryPoint, bool PackToFlash);
```

This function downloads a compiled C program to the memory of the specified **KMotion** board.

C Programs that run in the **KMotion** Board are normally compiled using the included and integrated compiler in the **KMotion** Application. Using the **KMotion** Application the user's C Program should be loaded into a selected thread and compiled. This will automatically generate a COFF executable with the same name and in the same directory as the C Source code, but with a .out extension. It is the users responsibility to keep track of which thread the COFF executable was compiled to execute in.

This function is used to download the COFF executable to the **KMotion's** memory. The entry point of the executable will be extracted from the file and returned to the caller. This entry point address should then be passed to the **KMotion** board and associated with the same thread that the file was compiled to execute in. The entry point should be passed using the "EntryPoint" command (See the example below).

The downloaded code may then be executed by issuing the "Execute" command

Return Value

returns 0 - if successful

Parameters

board

specifies which board in the system the command applies to

Name

Filename of coff file to download

EntryPoint

Address of the entry point of the C program extracted from the COFF file and returned to the caller

PackToFlash

Internal system command always specify as false

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM;
unsigned int EntryPoint;
if (KM.LoadCoff(0, "C:\\test.out", &EntryPoint, false))
return 1;

s.Format("EntryPoint 0 %08X", EntryPoint);
KM.WriteLine(0,s);

KM.WriteLine(0,"Execute 0");
```

int ServiceConsole(int board);

Services the ***KMotion Console*** data stream. The *Console* is a place where all unsolicited data, such as errors, or data "Printed" by user programs goes to. In between processing commands, ***KMotion*** uploads any unsolicited data it may have up to the host. The KMotionDLL driver buffers this data until some process declares itself as a *Console Handler* (See **SetConsoleCallback**) and makes calls to this function *ServiceConsole*.

This function should be called at regular intervals. If console data is available a call back to the Console Handler will

occur with one line of data.

Return Value

returns 0 - if successful

Parameters

board

specifies which board in the system the command applies to

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM;

int ConsoleHandler(const char *buf)
{
    MyLogData(buf);
    return 0;
}

KM.SetConsoleCallback(0, ConsoleHandler);
KM.ServiceConsole(0);
```

int SetConsoleCallback(int board, CONSOLE_HANDLER *ch);

Sets the user provided console callback function.

Return Value

returns 0 - if successful

Parameters

board

specifies which board in the system the command applies to

ch

name of console handler function

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM;

int ConsoleHandler(const char *buf)
{
    MyLogData(buf);
    return 0;
}
```

```
}  
  
KM.SetConsoleCallback(0, ConsoleHandler);  
KM.ServiceConsole(0);
```

```
int CompileAndLoadCoff(int board, const char *Name, int  
Thread);  
or  
int CompileAndLoadCoff(int board, const char *Name, int  
Thread, char *Err, int MaxErrLen);
```

Compiles the specified [C Program](#) file, downloads the object code to the specified Thread space, and sets the Entry Point, for the specified thread. Two versions of the function are supplied; one returns any error messages, the other does not.

The downloaded code may then be executed by issuing the [Execute](#) command.

Return Value

returns 0 - if successful

Parameters

board

specifies which board in the system the command applies to

Name

Filename of C Program to compile and download

Thread

Thread number where the program is to be compiled for and downloaded into. Valid range 1...7.

Err

Caller's supplied buffer for any error messages

MaxErrLen

Length of caller's supplied buffer for any error messages

Example

```
#include "KMotionDLL.h"
CKMotionDLL KMotion;
if (KM.CompileAndLoadCoff(0, "C:\\MyProgram.c", 1)
MyError();
if (KM.WriteLine(0, "Execute1") MyError();
```

```
int ListLocations(int *nlocations, int *list);
```

Returns the number of currently connected KMotion boards and a list of their USB location identifiers

Return Value

returns 0 - if successful

Parameters

nlocations

pointer to integer where the number of locations should be returned

List

pointer to array to be filled in with the list of USB location identifiers

Example

```
#include "KMotionDLL.h"
```

```
CKMotionDLL KM;  
int n_boards;  
int BoardList[256];  
if (KM.ListLocations(&n_boards, BoardList) MyError());
```

Script Commands

Commands (by category):

Parameters

[Accel <N>=<A>](#)
[BacklashAmount <N>=<A>](#)
[BacklashMode <N>=<M>](#)
[BacklashRate <N>=<R>](#)
[CommutationOffset <N>=<X>](#)
[D<N>=<M>](#)
[DeadBandGain<N>=<M>](#)
[DeadBandRange<N>=<M>](#)
[Dest<N>=<M>](#)
[FFAccel<N>=<M>](#)
[FFVel <N>=<M>](#)
[I<N>=<M>](#)
[IIR<N> <M>=<A1> <A2> <B0> <B1> <B2>](#)
[InputChan<M> <N>=<C>](#)
[InputGain<M> <N>=<G>](#)
[InputMode<N>=<M>](#)
[InputOffset<M> <N>=<O>](#)
[InvDistPerCycle<N>=<X>](#)
[Jerk<N>=<J>](#)
[Lead<N>=<M>](#)
[LimitSwitch<N>=<H>](#)
[MasterAxis<N>=<M>](#)
[MaxErr<N>=<M>](#)
[MaxFollowingError<N>=<M>](#)
[MaxI<N> <M>](#)
[MaxOutput<N>=<M>](#)
[OutputChan<M> <N>=<C>](#)
[OutputGain<N>=<G>](#)
[OutputOffset<N>=<O>](#)
[OutputMode<N>=<M>](#)
[P<N>=<M>](#)
[Pos<N>=<P>](#)
[SlaveGain<N>=<G>](#)
[StepperAmplitude<N>=<M>](#)
[Vel<N>=<V>](#)

User Threads

[EntryPoint<N> <H>](#)
[Execute<N>](#)
[Kill<N>](#)
[LoadData <H> <N>](#)
[SetStartupThread<N> <M>](#)

I/O Commands/Status

[ADC<N>](#)
[ClearBit<N>](#)
[ClearBitBuf<N>](#)
[DAC<N> <M>](#)
[GetBitDirection<N>](#)
[ReadBit<N>](#)
[SetBit<N>](#)
[SetBitBuf<N>](#)
[SetBitDirection<N>=<M>](#)
[SetStateBit<N>=<M>](#)
[SetStateBitBuf<N>=<M>](#)

Output Stage

[3PH<N>=<M> <A>](#)
[4PH<N>=<M> <A>](#)
[PWM<N>=<M>](#)
[PWMC<N>=<M>](#)
[PWMR<N>=<M>](#)

Gather Commands

[CheckDoneGather](#)
[GatherMove<N> <M> <L>](#)
[GatherStep<N> <M> <L>](#)
[GetGather <N>](#)
[GetGatherDec<N>](#)
[GetGatherHex<N> <M>](#)
[GetInject<N> <M>](#)
[Inject<N> <F> <A>](#)
[SetGatherDec <N> <M>](#)
[SetGatherHex<N> <M>](#)

FLASH Commands

[ClearFlashImage](#)
[Flash](#)
[LoadFlash<H> <N>](#)
[ProgFlashImage](#)

Motion Commands

[Arc <XC> <YC> <RX> <RY>](#)
[<θ0> <dθ> <Z0> <A0> <B0> <C0>](#)
[<Z1> <A1> <B1> <C1>](#)
[<a> <c> <d> <tF>](#)
[ArcXZ <XC> <ZC> <RX> <RZ>](#)
[<θ0> <dθ> <Y0> <A0> <B0> <C0>](#)
[<Y1> <A1> <B1> <C1>](#)
[<a> <c> <d> <tF>](#)
[ArcYZ <YC> <ZC> <RY> <RZ>](#)
[<θ0> <dθ> <X0> <A0> <B0> <C0>](#)
[<X1> <A1> <B1> <C1>](#)
[<a> <c> <d> <tF>](#)
[ArcHex <XC> <YC> <RX> <RY>](#)
[<θ0> <dθ> <Z0> <A0> <B0> <C0>](#)
[<Z1> <A1> <B1> <C1>](#)
[<a> <c> <d> <tF>](#)
[ArcHexXZ <XC> <ZC> <RX> <RZ>](#)
[<θ0> <dθ> <Y0> <A0> <B0> <C0>](#)
[<Y1> <A1> <B1> <C1>](#)
[<a> <c> <d> <tF>](#)
[ArcHexYZ <YC> <ZC> <RY> <RZ>](#)
[<θ0> <dθ> <X0> <A0> <B0> <C0>](#)
[<X1> <A1> <B1> <C1>](#)
[<a> <c> <d> <tF>](#)
[CheckDone<N>](#)
[CheckDoneBuf](#)
[CheckDoneXYZA](#)
[ConfigSpindle <T> <A> <U> <W> <C>](#)
[DefineCS <X> <Y> <Z> <A> <C>](#)
[DisableAxis<N>](#)
[EnableAxis<N>](#)
[EnableAxisDest<N> <M>](#)
[Enabled<N>](#)
[ExecBuf](#)
[ExecTime](#)
[GetSpindleRPS](#)
[Jog<N>=<V>](#)
[Linear <X0> <Y0> <Z0> <A0> <B0> <C0>](#)
[<X1> <Y1> <Z1> <A1> <B1> <C1>](#)
[<a> <c> <d> <tF>](#)
[LinearEx <X0> <Y0> <Z0> <A0> <B0> <C0>](#)
[<U0> <V0>](#)
[<X1> <Y1> <Z1> <A1> <B1> <C1> <U1> <V1>](#)
[<a> <c> <d> <tF>](#)
[LinearHex <X0> <Y0> <Z0> <A0> <B0> <C0>](#)
[<X1> <Y1> <Z1> <A1> <B1> <C1>](#)
[<a> <c> <d> <tF>](#)
[LinearHexEx <X0> <Y0> <Z0> <A0> <B0>](#)
[<C0> <U0> <V0>](#)
[<X1> <Y1> <Z1> <A1> <B1> <C1> <U1> <V1>](#)
[<a> <c> <d> <tF>](#)
[Move<N>=<M>](#)
[MoveAtVel<N>=<M> <V>](#)
[MoveRel<N>=<M>](#)
[MoveRelAtVel<N>=<M> <V>](#)
[MoveXYZA <X> <Y> <Z> <A>](#)
[OpenBuf](#)
[TrigThread <S>](#)
[Zero<N>](#)

Misc Commands

[Echo <S>](#)
[GetPersistDec<N>](#)
[GetPersistHex<N>](#)
[GetStatus](#)
[Reboot!](#)
[SetPersistDec <O> <D>](#)
[SetPersistHex <O> <H>](#)
[Version](#)

3PH<N>=<M> <A>**Description**

Sets the assigned PWMs of an axis to the specified magnitude and phase angle for a brushless 3 phase motor.

This command is useful for energizing a coil (or effective coil position). This is often required while initial homing or determining the commutation offset for a 3 phase brushless motor. If an effective coil position is energized, the motor rotor will normally align itself to the coil position. This is similar to the manner in which a stepping motor operates. Since the rotor location is then known, the commutation offset may then be determined. Alternately if an index mark is available, the effective coil position may be rotated by changing the phase angle until the index mark is detected.

Parameters**<N>**

Selected Axis for command. Valid range 0...7.

<M>

Magnitude of output to apply.

Valid Range is -230 ... +230 PWM units

<A>

Commutation angle to be used.

Units are in Commutation cycles

Only fractional value will be used

Example**3PH0=230 0.5**

4PH<N>=<M> <A>

Sets the assigned PWMs of an axis to the specified magnitude and phase angle for a brushless 4 phase motor.

This command is useful for energizing a coil (or effective coil position). This is often required while initial homing or determining the commutation offset for a 4 phase brushless motor. If an effective coil position is energized, the motor rotor will normally align itself to the coil position. This is similar to the manner in which a stepping motor operates. Since the rotor location is then known, the commutation offset may then be determined. Alternately if an index mark is available, the effective coil position may be rotated by changing the phase angle until the index mark is detected.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Magnitude of output to apply.

Valid Range is -250 ... +250 PWM units

<A>

Commutation angle to be used.

Units are in Commutation cycles

Only fractional value will be used

Example

4PH0=250 0.5

Accel <N>=<A>

or

Accel <N>

Description

Get or Set the max acceleration (for independent moves and jogs)

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<A>

The max acceleration. Units are in *Position units* per sec²

Example

Accel0=1000.0

ADC<N>

Description

Display current ADC (Analog to Digital Converter). Display range -2048 to 2047

Channels 0-3 are $\pm 10V$ general purpose inputs

Channels 4-7 are Motor Currents

Parameters

<N>

ADC channel

Valid range 0 ... 7

Example

ADC 0

**Arc <X_c> <Y_c> <R_x> <R_y> < θ_0 > <d θ > <Z₀> <A₀> <B₀> <C₀> <Z₁> <A₁> <B₁> <C₁> <a>
 <c> <d> <t_f>**

Description

Place circular (also elliptical or helical) interpolated move into the coordinated motion buffer. See also [KMotion Coordinated Motion](#). A path through space is defined where x and y are changing in an elliptical manner and z , a , b , c are changing in a linear manner forming a portion of a helix. A parametric equation is defined which describes which portion of the path as well as how as a function of time the path is to be traversed. This command can consist of up to 6 axis of coordinated motion. X and Y perform an arc while Z , A , B , and C move linearly.

Although the Arc command may be sent directly, the Arc command is normally generated automatically to perform a planned trajectory by the coordinated motion library or GCode.

(X_C, Y_C) - center of circle

(R_X, R_Y) - x radius and y radius

θ_0 - initial angle for the beginning of the path

$d\theta$ - amount of angular change for the path

Z_0 - initial Z position of path

A_0 - initial A position of path

B_0 - initial B position of path

C_0 - initial C position of path

Z_1 - final Z position of path

A_1 - final A position of path

B_1 - final B position of path

C_1 - final C position of path

3rd		order		parametric		equation		where			
p	=	a	t^3	+	b	t^2	+	c	t	+	d

p is the position along the path as a function of time. When $p=0$ the (x,y,z) position will be at the beginning of the path ($\theta = \theta_0$ and $z=z_0$). When $p=1$ the (x,y,z) position will be at the end of the path ($\theta = \theta_0 + d\theta$, and $z=z_1$).

This motion segment will be performed over a time period of t_F , where t varies from $0 \dots t_F$. Note that it is not necessary that p vary over the entire range of $0 \dots 1$. This is often the case when there may be an acceleration, constant velocity, and deceleration phase over the path. ie: t might vary from $0.0 \rightarrow 0.1$ where p might vary from $0.3 \rightarrow 0.7$.

Parameters

<X_c> - X center of ellipse, units are position units of x axis

<Y_c> - Y center of ellipse, units are position units of y axis

<R_x> - X radius of ellipse, units are position units of x axis

<R_y> - Y radius of ellipse, units are position units of y axis

<θ₀> - initial theta position on ellipse, radians (0 radians points in the +x direction)

<dθ> - change in theta position on ellipse, radians (+ theta causes CCW motion)

<Z₀> - initial Z position on path, units are position units of z axis

<A₀> - initial A position on path, units are position units of a axis

<B₀> - initial B position on path, units are position units of b axis

<C₀> - initial C position on path, units are position units of c axis

<Z₁> - final Z position on path, units are position units of z axis

<A₁> - final A position on path, units are position units of a axis

<B₁> - final B position on path, units are position units of b axis

<C₁> - final C position on path, units are position units of c axis

<a> - parametric equation t^3 coefficient

**** - parametric equation t^2 coefficient

<c> - parametric equation t coefficient

<d> - parametric equation constant coefficient

<t_f> - time for segment

Example (complete unit circle, centered at 0.5,0.5, no Z, A, B, or C motion, performed in 10 seconds)

Arc 0.5 0.5 1.0 1.0 0.0 6.28 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.1 0.0 10.0

ArcXZ <X_C> <Z_C> <R_X> <R_Z> <θ₀> <dθ> <Y₀> <A₀> <B₀> <C₀> <Y₁> <A₁> <B₁> <C₁>
<a> <c> <d> <t_F>

Description

Place circular (also elliptical or helical) interpolated move into the coordinated motion buffer. Same as [Arc](#) Command except circular motion is performed in the XZ plane rather than the XY plane.

ArcYZ <Y_C> <Z_C> <R_Y> <R_Z> <θ₀> <dθ> <X₀> <A₀> <B₀> <C₀> <X₁> <A₁> <B₁> <C₁>
<a> <c> <d> <t_F>

Description

Place circular (also elliptical or helical) interpolated move into the coordinated motion buffer. Same as [Arc](#) Command except circular motion is performed in the YZ plane rather than the XY plane.

ArcHex <X_C> <Y_C> <R_X> <R_Y> <θ₀> <dθ> <Z₀> <A₀> <B₀> <C₀> <Z₁> <A₁> <B₁> <C₁>
<a> <c> <d> <t_F>

Description

Place circular (also elliptical or helical) interpolated move into the coordinated motion buffer. This command is exactly the same as the [Arc](#) command above, except all 13 parameters are specified as 32-bit hexadecimal values which are the binary images of 32-bit floating point values. When generated by a program this is often faster, simpler, and more precise than decimal values. See also [KMotion Coordinated Motion](#).

Parameters

See above.

Example (complete unit circle, centered at 0.5,0.5, no Z motion, performed in 10 seconds)

Arc 3f000000 3f000000 3f800000 3f800000 0 40c8f5c3 0 0 0 40c8f5c3 0 41800000

ArcHexXZ <X_C> <Z_C> <R_X> <R_Z> <θ₀> <dθ> <Y₀> <A₀> <B₀> <C₀> <Y₁> <A₁> <B₁>
<C₁> <a> <c> <d> <t_F>

Description

Place circular (also elliptical or helical) interpolated move into the coordinated motion buffer. Same as [ArcHex](#) Command except circular motion is performed in the XZ plane rather than the XY plane.

ArcHexYZ <Y_C> <Z_C> <R_Y> <R_Z> <θ₀> <dθ> <X₀> <A₀> <B₀> <C₀> <X₁> <A₁> <B₁> <C₁> <a> <c> <d> <t_F>

Description

Place circular (also elliptical or helical) interpolated move into the coordinated motion buffer. Same as [ArcHex](#) Command except circular motion is performed in the YZ plane rather than the XY plane.

BacklashAmount<N>=<A>

or

BacklashAmount<N>

Description

Sets or gets the amount of Backlash Compensation Offset to be applied.

See also [BacklashMode](#) and [BacklashRate](#).

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<A>

Floating point Backlash Compensation Amount in units of Steps or Counts.

Example

BacklashAmount=15.5

BacklashMode<N>=<M>

or

BacklashMode<N>

Description

Sets or gets the Backlash Compensation mode from either BACKLASH_OFF (0) to BACKLASH_LINEAR (1). When the backlash mode is set to Linear mode, whenever the commanded destination begins moving in the positive direction, a positive offset of the amount, [BacklashAmount](#), will be applied. The offset will be ramped upward as a linear function of time at the rate specified as the [BacklashRate](#). Whenever the commanded destination begins moving in the negative direction the offset will be removed by ramping downward toward zero at the same rate.

If the the Backlash Compensation mode is set to BACKLASH_OFF (0), no backlash compensation will be applied.

Parameters**<N>**

Selected Axis for command. Valid range 0...7.

<M>

Backlash Compensation Mode setting. Currently 0 or 1.

Example**BacklashMode0=1**

BacklashRate<N>=<R>

or

BacklashRate<N>**Description**

Sets or gets the rate at which the amount of Backlash Compensation Offset will be applied.

See also [BacklashMode](#) and [BacklashAmount](#).

Parameters**<N>**

Selected Axis for command. Valid range 0...7.

<A>

Floating point Backlash Compensation Rate in units of Steps or Counts per second.

Example

BacklashRate=1000.0

CheckDone<N>

Description

Displays:

1 if axis N has completed its motion

0 if axis N has not completed its motion

-1 if the axis is disabled

Parameters

<N>

Selected Axis for command. Valid range 0...7.

Example

CheckDone0

CheckDoneBuf

Description

Displays the status of the Coordinated Motion Buffer. **KMotion** contains a Coordinated Motion Buffer where move segments (linear and arcs) and I/O commands may be downloaded and executed in real time.

Displays:

1 if all coordinated move segments have completed

0 if all coordinated move segments have not completed

-1 if any axis in the [defined coordinate system](#) is disabled

Parameters

None

Example

CheckDoneBuf

CheckDoneGather

Description

Displays the status of a data gather operation. **KMotion** contains a mechanism for capturing data from a variety of sources in real time. This mechanism is utilized when capturing data for [Code plots](#) and [Step response](#) plots. It is also available for general purpose use. See the [data gathering example](#).

Displays:

1 if data gather is completed

0 if data gather has not completed

Parameters

None

Example

CheckDoneGather

CheckDoneXYZABC

Description

Displays status of a commanded [MoveXYZABC](#) command. See also [DefineCS6](#).

Displays:

1 if all axes in the defined coordinate system have completed their motion

0 if any axis in the defined coordinate system has not completed its motion

-1 if any axis in the defined coordinate system is disabled

Parameters

None

Example

CheckDoneXYZABC

ClearBit<N>**Description**

Clears an actual I/O bit or virtual I/O bit. Note that actual IO bits must be previously defined as an output, see [SetBitDirection](#)

Parameters

<N>

Bit number specified as a decimal number. Valid range 0...31 for actual hardware I/O bits. Valid range of 32...63 for virtual I/O bits.

Example

ClearBit0

ClearBitBuf<N>**Description**

Inserts into the coordinated move buffer a command to clear an IO bit N(0..30) or a Virtual IO bit (32..63) (actual IO bits must be defined as an output, see [SetBitDirection](#))

Parameters

<N>

Bit Number to clear. Valid Range 0...63.

Example

ClearBitBuf0

ClearFlashImage**Description**

Prepare to download FLASH firmware image. Sets entire RAM flash image to zero

Parameters

None.

Example

ClearFlashImage

CommutationOffset<N>=<X>

or

CommutationOffset<N>**Description**

Get or Set 3 or 4 phase commutation offset. When brushless commutation is performed, the desired Output Magnitude is distributed and applied to the various motor coils as a function of position. The commutation offset shifts the manner in which the Output Magnitude is applied.

For a 3 phase brushless output mode, commutation offset is used in the following manner.

$$\text{PhaseA} = \text{OutputMagnitude} * \sin((\text{Position} + \text{CommutationOffset}) * \text{invDistPerCycle} * 2\pi)$$

$$\text{PhaseB} = \text{OutputMagnitude} * \sin((\text{Position} + \text{CommutationOffset}) * \text{invDistPerCycle} * 2\pi + 2\pi/3)$$

$$\text{PhaseC} = \text{OutputMagnitude} * \sin((\text{Position} + \text{CommutationOffset}) * \text{invDistPerCycle} * 2\pi + 4\pi/3)$$

For a 4 phase brushless output mode, commutation offset is used in the following manner.

$$\text{PhaseA} = \text{OutputMagnitude} * \sin((\text{Position} + \text{CommutationOffset}) * \text{invDistPerCycle} * 2\pi)$$

$$\text{PhaseB} = \text{OutputMagnitude} * \cos((\text{Position} + \text{CommutationOffset}) * \text{invDistPerCycle} * 2\pi)$$

See also [invDistPerCycle](#) and [Configuration Parameters](#).

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<X>

Offset in units of Position.

Example

CommutationOffset0=100.0

ConfigSpindle <T> <A> <U> <W> <C>**Description**

Enables/Disables and configures the firmware to monitor Spindle Speed and Position to allow reporting of Spindle Speed and to perform Threading operations.

See also: [TrigThread](#) and [GetSpindleRPS](#)

Parameters**<T>**

Spindle Sensor Type. 0 - disables spindle measurement, 1 - defines the sensor type as a quadrature encoder .

<A>

Axis - Defines the Axis Channel that will maintain the Spindle Position. Note this is not a Encoder input channel. Rather it is the Axis Channel that has a Encoder input Channel defined. Valid range 0 ...7.

<U>

Update Time - delta time for measurement. This is the amount of time between Spindle Position samples used to calculate the current speed. $Speed = \Delta Position / \Delta Time$. A longer time period will allow for a more accurate speed measurement, especially at low speeds and if a low resolution encoder is used. A shorter Update Time will make the speed measurement to be more responsive as it changes. Units of seconds. Typical value 0.2 seconds

<W>

Tau - low pass filter time constant for threading. Pseudo Time along a time dependent trajectory path is adjusted based on spindle position. The Pseudo Time is smoothed using a low pass filter with a time constant of Tau to avoid making too abrupt changes of position, velocity or acceleration. Units of seconds. Typical value 0.1 seconds

<C>

Counts per Revolution. Number of encoder counts per full revolution of the Spindle.

Example

ConfigureSpindle 1 0 0.2 0.1 4096.0

D<N>=<M>

or

D<N>

Description

Get or Set [PID](#) derivative Gain.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Derivative Gain value. The units of the derivative gain are in Output Units/Position Units x [Servo Sample Time](#).

Example

D0=10.0

DAC<N> <M>

Description

DAC to value. DACs 0...3 have ± 10 Volt ranges, DACs 4...7 have 0...4 Volt ranges. See also [Analog Status Screen](#).

Parameters

<N>

DAC channel to set. Valid Range 0...7.

<M>

DAC value to set in counts. Valid Range -2048...2047.

Example

DAC0=2000

DeadBandGain<N>=<M>

or

DeadBandGain<N>

Description

Get or Set gain while error is within the [deadband range](#). See [DeadBand Description](#). See [Servo Flow Diagram](#).

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Gain to be applied. A value of 1.0 will have normal gain while within the deadband. A value less than 1.0 will have reduced gain within the deadband.

Example

DeadBandGain0=0.5

DeadBandRange<N>=<M>

or

DeadBandRange<N>

Description

Get or Set range where [deadband gain](#) is to be applied. See [DeadBand Description](#). See [Servo Flow Diagram](#).

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

±Range in Position units,

Example

DeadBandRange0=1.0

DefineCS<X> <Y> <Z> <A> <C>

or

DefineCS

Description

Set or get the defined X Y Z A B C coordinate system axis assignments for up to 6 axes of coordinated motion. Unused axis are assigned an axis channel of -1.

See also [Coordinated Motion](#).

Parameters

<X>

Assigned Axis channel number for X. Valid range -1 ... 7.

Use -1 if axis is not defined.

<Y>

Assigned Axis channel number for Y. Valid range -1 ... 7.

Use -1 if axis is not defined.

<Z>

Assigned Axis channel number for Z. Valid range -1 ... 7.

Use -1 if axis is not defined.

<A>

Assigned Axis channel number for A. Valid range -1 ... 7.

Use -1 if axis is not defined.

Assigned Axis channel number for B. Valid range -1 ... 7.

Use -1 if axis is not defined.

<C>

Assigned Axis channel number for C. Valid range -1 ... 7.

Use -1 if axis is not defined.

Example

DefineCS

DefineCS = 0 1 2 3 4 -1

Dest<N>=<M>

or

Dest<N>

Description

Set or get the last commanded destination for an axis. The Dest (destination) is normally set (or continuously updated) as the result of a motion command (Move, Jog, or Coordinated motion) , but may also be set with this command if no motion is in progress.

Parameters**<N>**

Selected Axis for command. Valid range 0...7.

<M>

Value to set in Position units. Valid range - any.

Example**Dest0=100****or****Dest0**

DisableAxis<N>**Description**

Kill any motion and disable motor. Any associated output PWM channels for the axis will be set to 0R mode.

Parameters**<N>**

Selected Axis for command. Valid range 0...7.

Example**DisableAxis0**

Echo <S>**Description**

Echo character string back to the [Console Screen](#).

Parameters**<S>**

Any character string < 80 characters

Example

Echo Hello

EnableAxis<N>

Description

Set an Axis' *destination* to the Current Measured Position and enable the axis. See also [EnableAxisDest](#) to explicitly set the desired destination for the axis. Note for a MicroStepper Axis (which normally has no measured position) this command will leave the Axis' destination unchanged. .

Parameters

<N>

Selected Axis for command. Valid range 0...7.

Example

Enable0

EnableAxisDest<N> <M>

Description

Set an Axis' *destination* to the specified position and enable the axis. See also [EnableAxis](#) to set the desired destination to the current measured position.

<N>

Selected Axis for command. Valid range 0...7.

<M>

Destination for the axis. Position units. Valid range - any.

Example

EnableAxisDest0 1000.0

Enabled<N>**Description**

Display whether the specified axis is enabled, 1 - if currently enabled, 0 - if not enabled.

Note: to enable an axis use [EnableAxis](#) or [EnableAxisDest](#).

Parameters

<N>

Selected Axis for command. Valid range 0...7.

Example

Enabled0

EntryPoint<N> <H>**Description**

Set execution start address of user thread to specified address. This operation is normally performed automatically when downloading a user program.

Parameters

<N>

User Thread number to set. Decimal number. Valid range 1...7.

<H>

Start address. 32 bit Hex number.

Example

Entrypoint1 80030000

ExecBuf**Description**

Execute the contents of the coordinated motion buffer. Use [CheckDoneBuf](#) to determine when the buffer has been fully executed. See also [Coordinated Motion](#).

Parameters

None

Example

ExecBuf

ExecTime

Description

Displays the amount of the Coordinated Motion Buffer that has been already executed in terms of *Time*. **KMotion** contains a Coordinated Motion Buffer where move segments (linear and arcs) and I/O commands may be downloaded and executed in real time. This command is useful for determining how long before the Coordinated Motion Buffer will complete. For example, if a number of segments have been downloaded where their total execution time is 10 seconds, and they are currently in progress of being executed, and the ExecTime command reports that 8 seconds worth of segments have been executed, then the remaining time before the queue completes (or starves for data) would be 2 seconds. This command is useful for applications where it is important not to download data too far ahead so changes to the Trajectory may be made. The value returned is a floating point decimal value in Seconds with 3 decimal places. If the Coordinated Motion has already completed the amount of time will be a negative value whose magnitude is the total time that was executed. See also [Coordinated Motion](#).

Displays:

Executed time in seconds as a floating point decimal number with 3 decimal places

ie. 10.123

If the buffer has already completed the value will be negative

ie. -10.123

Parameters

None

Example

ExecTime

Execute<N>**Description**

Begin execution of thread. Execution begins at the previously specified thread entry point.

See also [C Program Screen](#).

Parameters

<N>

Thread number to begin execution. Decimal number. Valid range 1...7.

Example

Execute1

FFAccel<N>=<M>

or

FFAccel<N>**Description**

Set or get Acceleration *feed forward* for axis.

See also [feed forward tuning](#).

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Feed forward value. units are in Output units per Input Units per sec².

Example

FFAccel0=100.0

or

FFAccel0

FFVel<N>=<M>

or

FFVel<N>

Description

Set or get Velocity *feed forward* for axis.

See also [feed forward tuning](#).

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Feed forward value. units are in Output units per Input Units per sec.

Example

FFVel0=100.0

or

FFVel0

Flash

Description

Flash current *user programs*, persistent memory area, all axes configurations, tuning, and filter parameters to non-volatile memory. The entire state of the **KMotion** is saved to FLASH memory. Any active user programs will be paused during the flash operation

Parameters

None

Example

Flash

GatherMove<N> <M> <L>

Description

Performs a profiled move on an axis of the specified distance while [gathering](#) the specified number of points of data. This command is used while gathering data for the [Step Response Screen](#) plots.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Distance to move. Units are Position Units. Valid Range - any.

<L>

Number of servo samples to gather. Valid Range - 1...40000

Example

GatherMove0 1000.0 2000

GatherStep<N> <M> <L>

Description

Performs a step on an axis of the specified distance while [gathering](#) the specified number of points of data. This command is used while gathering data for the [Step Response Screen](#) plots.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Distance to step. Units are Position Units. Valid Range - any.

<L>

Number of servo samples to gather. Valid Range - 1...40000

Example

GatherStep0 1000.0 2000

GetBitDirection<N>

Description

Displays whether an IO bit N (0..30) is defined as input (0) or output (1)

Parameters

<N>

I/O bit number. Valid range 0...30

Example

GetBitDirection0

GetGather <N>

Description

Upload N data points from previous [GatherMove](#) or [GatherStep](#) command. Captured commanded destination, measured position, and output are uploaded as hex values (that represent binary images of 32-bit floating point values). Eight samples (24 values) per line.

Parameters

<N>

Number of points to upload. Valid range 1...40000.

Example

GetGather 1000

GetGatherDec<N>**Description**

Reads a single word from the [Gather Buffer](#) at the specified offset. A single 32-bit value displayed as a signed decimal integer number will be displayed.

Parameters

<N>

Offset into gather buffer, specified as a decimal offset of 32 bit words. Valid range 0...1999999

Example

GetGatherDec 1000

GetGatherHex<N> <M>**Description**

Reads multiple words from the [Gather Buffer](#) beginning at the specified offset. Hexadecimal values will be displayed that will represent binary images of the contents of the gather buffer as 32 bit words.

Parameters

<N>

Offset into gather buffer, specified as a decimal offset of 32 bit words. Valid range 0...1999999

<M>

Number of 32 bit words to display. Decimal integer. Valid range 1...2000000

Example

GetGatherHex 0 100

GetInject<N> <M>**Description**

Display results of signal injection and gathering. Bode Plot measurement involves injecting a signal and measuring the response for each of N_CPLX (2048) samples. This command gets the result from the injection. 3 values per sample are uploaded. Injection value, position response (relative to destination), and servo output. All 3 values are printed as hexadecimal values which represent the image of a 32-bit floating point value. 8 samples (24 hex values) are printed per line.

Parameters

None

Example

GetInject

GetPersistDec<N>

Description

Read a single word from the *Persist Array* at the specified offset a single 32-bit value displayed as a **signed decimal number**. The persist array is a general purpose array of N_USER_DATA_VARS (100) 32-bit values that is accessible to the host as well as **KMotion C Programs**. It may be used to share parameters, commands, or information between programs.

C Programs may access this array as the integer array:

```
persist.UserData[n] ;
```

It also resides in the **KMotion** Persist memory structure so that if memory is flashed, the value will be present at power up.

See also [GetPersistHex](#), [SetPersistDec](#), [SetPersistHex](#)

Parameters

<N>

Offset into the integer array. Valid range 0...99.

Example

GetPersistDec 10

GetPersistHex<N>

Description

Read a single word from the *Persist Array* at the specified offset a single 32-bit value displayed as an **unsigned hexadecimal number**. The persist array is a general purpose array of N_USER_DATA_VARS (100) 32-bit values that is accessible to the host as well as **KMotion C Programs**. It may be used to share parameters, commands, or information between programs.

C Programs may access this array as the integer array:

```
persist.UserData[n] ;
```

It also resides in the **KMotion** Persist memory structure so that if memory is flashed, the value will be present at power up.

See also [GetPersistDec](#), [SetPersistDec](#), [SetPersistHex](#)

Parameters

<N>

Offset into the integer array. Valid range 0...99.

Example

GetPersistHex 10

GetSpindleRPS

Description

Reports the current Spindle Speed in revolutions per second.

See also [ConfigSpindle](#) and [TrigThread](#)

Parameters

Example

GetSpindleRPS

GetStatus

Description

Upload Main Status record in hex format. **KMotion** provides a means of quickly uploading the most commonly used status. This information is defined in the PC-DSP.h header file as the MAIN_STATUS structure. The entire structure is uploaded as a binary image represented as 32-bit hexadecimal values.

Parameters

None

Example

GetStatus

I<N>=<M>

or

I<N>**Description**

Get or Set [PID](#) Integral Gain.

Parameters**<N>**

Selected Axis for command. Valid range 0...7.

<M>

Integral Gain value. The units of the derivative gain are in Output Units x Position Units x [Servo Sample Time](#).

Example**I0=10.0**

or

I0

IIR<N> <M>=<A1> <A2> <B0> <B1> <B2>

or

IIR<N> <M>

Description

Set or get IIR Z domain servo filter.

See also [IIR Filter Screen](#)

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Filter number for axis. Valid range 0...2.

<A1> <A2> <B0> <B1> <B2>

Filter coefficients represented as floating point decimal values.

Example

IIR0 0=1.5 2.5 -3.5 4.5 5.5

or

IIR0 0

Inject<N> <F> <A>

Description

A Inject random stimulus into an axis with the specified cutoff frequency and amplitude. Useful for generating [Bode plots](#).

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<F>

Cutoff Frequency in Hz. Valid range - any.

<A>

Amplitude in position units. Valid range - any.

Example

Inject0 100.0 20.0

InputChan<M> <N>=<C>

or

InputChan<M> <N>

Description

Get or Set the first or second *Input Channel* of an axis. See description of this parameter on the [Configuration Screen](#).

Parameters

<M>

Selected input channel. Valid range 0...1.

<N>

Selected Axis for command. Valid range 0...7.

<C>

Channel number to assign. Valid range 0...7.

Example (set first input channel of axis 3 to 3)

InputChan0 3=3

or

InputChan0 3

InputGain<M> <N>=<G>

or

InputGain<M> <N>

Description

Set or get first or second Input Gain of an axis. See description of this parameter on the [Configuration Screen](#).

Parameters

<M>

Selected input channel. Valid range 0...1.

<N>

Selected Axis for command. Valid range 0...7.

<C>

Input Gain. Valid range - any.

Example

InputGain0 3=1.0

InputMode<N>=<M>

or

InputMode<N>

Description

Set or get the position input mode for an axis. See description of this parameter on the [Configuration Screen](#).

Valid modes are:

ENCODER_MODE 1

ADC_MODE 2

RESOLVER_MODE 3

USER_INPUT_MODE 4

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Mode. Valid range 1...4

Example

SetInputMode0=1

InputOffset<M> <N>=<O>

or

InputOffset<M> <N>

Description

Set or get first or second Input Offset of an axis. See description of this parameter on the [Configuration Screen](#).

Parameters

<M>

Selected input channel. Valid range 0...1.

<N>

Selected Axis for command. Valid range 0...7.

<O>

Input Offset. Valid range - any.

Example

InputOffset0 3=0.0

InvDistPerCycle<N>=<X>

Description

Get or Set distance per cycle (specified as an inverse) of an axis. May specify the cycle of either a Stepper or Brushless Motor.

See description of this parameter on the [Configuration Screen](#).

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<X>

Inverse (reciprocal) of distance for a complete cycle. Inverse position units. Should be specified exactly or with very high precision (double precision accuracy ~ 15 digits). Valid range - any.

Example

InvDistPerCycle0=0.05

Jerk<N>=<J>

or

Jerk<N>

Description

Get or Set the max [jerk](#) (for independent moves and jogs)

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<J>

The max Jerk. Units are in *Position units* per sec³

Example

Jerk0=10000.0

Jog<N>=<V>

Description

Move at constant velocity. Uses [Accel](#) and [Jerk](#) parameters for the axis to accelerate from the current velocity to the specified velocity. Axis should be already enabled. Specify zero velocity to decelerate to a stop.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<V>

new Velocity in position units/second. Valid range - any.

Example

Jog0=-200.5

Kill<N>

Description

Stop execution of a user thread.

Parameters

<N>

Thread to halt. Valid range 1..7

Example

Kill0

Lead<N>=<M>

or

Lead<N>***Description***

Set or get [Lead Compensation](#) for an axis. Lead Compensation is used to compensate for lag caused by motor inductance.

Parameters**<N>**

Selected Axis for command. Valid range 0...7.

<M>

Lead Compensation. Valid range - any.

Example**Lead0=10.0**

or

Lead0

LimitSwitch<N>=<H>***Description***

Configures Limit Switch Options. Specify Hex value where:

See also [Configuration Screen](#).

Parameters**<N>**

Selected Axis for command. Valid range 0...7.

<H>

32-bit hexadecimal value:

Bit 0 1=Stop Motor on Neg Limit, 0=Ignore Neg limit

Bit 1 1=Stop Motor on Pos Limit, 0=Ignore Pos limit

Bit 2 Neg Limit Polarity 0=stop on high, 1=stop on low

Bit 3 Pos Limit Polarity 0=stop on high, 1=stop on low

Bits 4-7 Action - 0 Kill Motor Drive

1 Disallow drive in direction of limit

2 Stop movement

Bits 16-23 Neg Limit Bit number

Bits 24-31 Pos Limit Bit number

Example

LimitSwitch2 0C0D0003

**Linear <X₀> <Y₀> <Z₀> <A₀> <B₀> <C₀> <X₁> <Y₁> <Z₁> <A₁> <B₁> <C₁> <a> <c>
<d> <t_F>**

Description

Place linear (in 6 dimensions) interpolated move into the coordinated motion buffer. See also [KMotion Coordinated Motion](#). A path through space is defined where x , y , z , a , b , and c are changing in a linear manner. A parametric equation is defined which describes which portion of the path as well as how as a function of time the path is to be traversed.

Although the Linear command may be sent directly, the Linear command is normally generated automatically to perform a planned trajectory by the coordinated motion library or GCode.

$(X_0, Y_0, Z_0, A_0, B_0, C_0)$ - beginning of path

$(X_1, Y_1, Z_1, A_1, B_1, C_1)$ - end of path

3rd order parametric equation where

$$p = a t^3 + b t^2 + c t + d$$

p is the position along the path as a function of time. When $p=0$ the (x,y,z,A) position will be at the beginning of the path. When $p=1$ the (x,y,z,A) position will be at the end of the path.

This motion segment will be performed over a time period of t_F , where t varies from 0 ... t_F . Note that it is not necessary that p vary over the entire range of 0 ... 1. This is often the case when there may be an acceleration, constant velocity, and deceleration phase over the path. ie: t might vary from 0.0->0.1 where p might vary from 0.3->0.7.

Parameters

< X_0 > - X begin point

< Y_0 > - Y begin point

< Z_0 > - Z begin point

< A_0 > - A begin point

< B_0 > - B begin point

< C_0 > - C begin point

< X_1 > - X end point

< Y_1 > - Y end point

< Z_1 > - Z end point

< A_1 > - A end point

< B_1 > - B end point

< C_1 > - C end point

< θ_1 > - initial theta position on ellipse, radians (0 radians points in the +x direction)

< a > - parametric equation t^3 coefficient

< b > - parametric equation t^2 coefficient

< c > - parametric equation t coefficient

< d > - parametric equation constant coefficient

< t_F > - time for segment

Example

Linear 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 1.0 1.0 1.0 1.0 0.0 0.0 1.0 0.0 1.0

LinearEx <X₀> <Y₀> <Z₀> <A₀> <B₀> <C₀> <U₀> <V₀> <X₁> <Y₁> <Z₁> <A₁> <B₁> <C₁>
<U₁> <V₁> <a> <c> <d> <t_F>

Description

Place linear (in 8 dimensions) interpolated move into the coordinated motion buffer. See also [KMotion Coordinated Motion](#). A path through space is defined where x, y, z, a, b, c, u and v are changing in a linear manner. A parametric equation is defined which describes which portion of the path as well as how as a function of time the path is to be traversed.

Although the Linear command may be sent directly, the Linear command is normally generated automatically to perform a planned trajectory by the coordinated motion library or GCode, however currently the GCode Interpreters available only support 6 axes of simultaneous motion.

(X₀, Y₀, Z₀, A₀, B₀, C₀, U₀, V₀) - beginning of path

(X₁, Y₁, Z₁, A₁, B₁, C₁, U₁, V₁) - end of path

3rd		order		parametric		equation		where			
p	=	a	t^3	+	b	t^2	+	c	t	+	d

p is the position along the path as a function of time. When $p=0$ the (x,y,z,A) position will be at the beginning of the path. When $p=1$ the (x,y,z,A) position will be at the end of the path.

This motion segment will be performed over a time period of t_F , where t varies from 0 ... t_F . Note that it is not necessary that p vary over the entire range of 0 ... 1. This is often the case when there may be an acceleration, constant velocity, and deceleration phase over the path. ie: t might vary from 0.0->0.1 where p might vary from 0.3->0.7.

Parameters

<X₀> - X begin point

<Y₀> - Y begin point

<Z₀> - Z begin point

<A₀> - A begin point

<B₀> - B begin point

<**C₀**> - C begin point

<**U₀**> - U begin point

<**V₀**> - V begin point

<**X₀**> - X end point

<**Y₁**> - Y end point

<**Z₁**> - Z end point

<**A₁**> - A end point

<**B₁**> - B end point

<**C₁**> - C end point

<**U₁**> - U end point

<**V₁**> - V end point

< **θ_1** > - initial theta position on ellipse, radians (0 radians points in the +x direction)

<**a**> - parametric equation t^3 coefficient

<**b**> - parametric equation t^2 coefficient

<**c**> - parametric equation t coefficient

<**d**> - parametric equation constant coefficient

<**t_F**> - time for segment

Example

LinearEx 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.0 0.0 1.0 0.0 1.0

**LinearHex <X₀> <Y₀> <Z₀> <A₀> <B₀> <C₀> <X₁> <Y₁> <Z₁> <A₁> <B₁> <C₁> <a>
<c> <d> <t_F>**

Description

Place linear (in 6 dimensions) interpolated move into the coordinated motion buffer. This command is exactly the same as the [Linear](#) command above, except all 17 parameters are specified as 32-bit hexadecimal values which are the binary images of 32-bit floating point values. When generated by

a program this is often faster, simpler, and more precise than decimal values. See also [KMotion Coordinated Motion](#).

Parameters

See above.

Example

```
LinearHex 0 0 0 0 0 0 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 0 0
3F800000 0 3F800000
```

```
LinearHexEx <X0> <Y0> <Z0> <A0> <B0> <C0> <U0> <V0> <X1> <Y1> <Z1> <A1> <B1>
<C1> <U1> <V1> <a> <b> <c> <d> <tf>
```

Description

Place linear (in 8 dimensions) interpolated move into the coordinated motion buffer. This command is exactly the same as the [LinearEx](#) command above, except all 21 parameters are specified as 32-bit hexadecimal values which are the binary images of 32-bit floating point values. When generated by a program this is often faster, simpler, and more precise than decimal values. See also [KMotion Coordinated Motion](#).

Parameters

See above.

Example

```
LinearHex 0 0 0 0 0 0 0 0 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000
3F800000 3F800000 0 0 3F800000 0 3F800000
```

```
LoadData <H> <N>
```

```
<B> <B> <B> <B> <B> ...
```

Description

Store data bytes into memory beginning at specified address for N bytes. The data must follow with up to N_BYTES_PER_LINE (64) bytes per line. This command is normally only used by the [COFF](#) loader. Since this command spans several lines, it may only be used programatically in conjunction with a [KMotionLock](#) or [WaitToken](#) command so that it is not interrupted.

Parameters

<H>

32-bit hexadecimal address

<N>

Number of bytes to follow and to be stored

** ...**

Bytes to store. 2 hexadecimal digits per byte, separated with a space.

Example

LoadData 80030000 4

FF FF FF FF

LoadFlash<H> <N>

** ...**

Description

Store data into FLASH image. Only by **KMotion** for downloading a new firmware version. Store data bytes into memory beginning at specified address for N bytes. The data must follow with up to N_BYTES_PER_LINE (64) bytes per line. This command is normally only used by the [COFF](#) loader. Since this command spans several lines, it may only be used programmatically in conjunction with a [KMotionLock](#) or [WaitToken](#) command so that it is not interrupted.

Parameters

<H>

32-bit hexadecimal address

<N>

Number of bytes to follow and to be stored

** ...**

Bytes to store. 2 hexadecimal digits per byte, separated with a space.

Example

LoadFlash FF00 4

FF FF FF FF

MasterAxis<N>=<M>

or

MasterAxis<N>

Description

Sets or gets the axis <M> to which the current axis <N> is to be slaved. The current axis becomes a slave and will follow the motion of the specified Master Axis. More than one axis can be slaved to a single master axis if desired. When slaved, changes in the commanded destination of the master axis will be mirrored as changes in the slaved axis's destination however scaled by the [SlaveGain](#) (as specified in the Slave Axis). The SlaveGain may be negative if opposing motion is desired.

Setting the Master Axis value to -1 disables the Slave mode.

Parameters

<N>

Selected Axis for command. Valid range 0 ... 7.

<M>

Master Axis or -1 to disable. Valid range -1 ... 7.

Example (set axis 1 to follow axis 0)

MasterAxis1=0

or

MasterAxis

MaxErr<N>=<M>

or

MaxErr<N>**Description**

Set or get *Maximum Error* for axis (Limits magnitude of error entering PID).

See [Servo Flow Diagram](#) and [Step Response Screen](#) for more information.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Maximum Error. Valid range - any positive value. Set to a [large value](#) to disable.

Example

MaxErr0=100.0

or

MaxErr0

MaxFollowingError<N>=<M>

or

MaxFollowingError<N>**Description**

Set or get the maximum allowed [following error](#) before [disabling](#) the axis.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Maximum Following Error. Valid range - any positive value. Set to a [large value](#) to disable.

Example

MaxFollowingError0=100.0

or

MaxFollowingError0

MaxI<N> <M>

Description

Set or get Maximum Integrator "wind up" for axis. Integrator saturates at the specified value.

See also [Servo Flow Diagram](#) and [Step Response Screen](#) for further information.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Maximum Integrator value. Valid range - any positive value. Set to a [large value](#) to disable.

Example

MaxI0=100.0

or

MaxI0

MaxOutput<N>=<M>

or

MaxOutput<N>

Description

Set or get Maximum Output for an axis. Limits magnitude of servo output. Output saturates at the specified value.

See also [Servo Flow Diagram](#) and [Step Response Screen](#) for further information.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Maximum output value. Valid range - any positive value. Set to a [large value](#) to disable.

Example

MaxOutput0=100.0

or

MaxOutput

Move<N>=<M>

Description

Move axis to absolute position. Axis should be already enabled. Uses [Vel](#), [Accel](#) and [Jerk](#) parameters for the axis to profile a motion from the current state to the specified position.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

new *position* in position units. Valid range - any.

Example

Move0=100.1

MoveAtVel<N>=<M> <V>**Description**

Move axis to absolute position at the specified Velocity. Axis should be already enabled. Uses [Accel](#) and [Jerk](#) parameters for the axis to profile a motion from the current state to the specified position.

Parameters**<N>**

Selected Axis for command. Valid range 0...7.

<M>

new *position* in position units. Valid range - any.

<V>

Desired Velocity for the Motion. Valid range - any.

Example**MoveAtVel0=100.1 30.0**

MoveRel<N>=<M>**Description**

Move axis relative to current destination. Same as [Move](#) command except specified motion is relative to current destination.

Axis should be already enabled. Uses [Vel](#), [Accel](#) and [Jerk](#) parameters for the axis to profile a motion from the current state to the specified position.

Parameters**<N>**

Selected Axis for command. Valid range 0...7.

<M>

Distance to move in position units. Valid range - any.

Example

MoveRel0=100.1

MoveRelAtVel<N>=<M> <V>

Description

Move axis relative to current destination at the specified Velocity. Same as [MoveAtVel](#) command except specified motion is relative to current destination. Axis should be already enabled. Uses [Accel](#) and [Jerk](#) parameters for the axis to profile a motion from the current state to the specified position.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

new *position* in position units. Valid range - any.

<V>

Desired Velocity for the Motion. Valid range - any.

Example

MoveRelAtVel0=100.1 30.0

MoveXYZABC <X> <Y> <Z> <A> <C>

Description

Move the 4 axes defined to be x,y,z,A (each axis moves independently). The [defined coordinate system](#) determines which axes channels are commanded to move.

Parameters

<X>

Position to move x axis. Valid range - any.

<Y>

Position to move y axis. Valid range - any.

<Z>

Position to move z axis. Valid range - any.

<A>

Position to move a axis. Valid range - any.

Position to move b axis. Valid range - any.

<C>

Position to move c axis. Valid range - any.

Example

MoveXYZABC 100.1 200.2 300.3 400.4 500.5 600.6

OpenBuf

Description

Clear and open the buffer for [coordinated motion](#).

Parameters

None

Example

OpenBuf

OutputChan<M> <N>=<C>

or

OutputChan<M> <N>

Description

Get or Set the first or second *Output Channel* of an axis. See description of this parameter on the [Configuration Screen](#).

Parameters

<M>

Selected input channel. Valid range 0...1.

<N>

Selected Axis for command. Valid range 0...7.

<C>

Channel number to assign. Valid range 0...7.

Example (set first output channel of axis 3 to 3)

OutputChan03=3

OutputGain<N>=<G>

or

OutputGain<N>

Description

Get or Set the *Output Gain* of an axis. For Axes of Step/Dir, CL Step Dir, or MicroStep output mode, the output motion can be scaled or reversed. Normally there is no need to use a value other than -1.0 or +1.0. For DAC Servo output mode the output signal (DAC) can be scaled or reversed. Again, normally there is no need to use a value other than -1.0 or +1.0. In other output modes the OutputGain value will have no effect.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<G>

Gain value. Valid range any floating point value.

Example

OutputGain0=-1.0

or

OutputGain0

OutputOffset<N>=<O>

or

OutputOffset<N>

Description

Get or Set the *Output Offset* of an axis. For DAC Servo output mode the output (DAC) signal can be offset. The Output Offset is applied after any [Output Gain](#) value. The Output Offset can be used to reduce any DAC output offset or Amplifier input offset that may cause motor axis drift occurs when the DAC is commanded to zero (disabled). In other output modes the OutputGain value will have no effect.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<G>

Gain value. Valid range any floating point value.

Example

OutputGain0=-1.0

or

OutputGain0

OutputMode<N>=<M>

or

OutputMode<N>

Description

Set or get the position output mode for an axis. See description of this parameter on the [Configuration Screen](#).

Valid modes are:

MICROSTEP_MODE 1

DC_SERVO_MODE 2

BRUSHLESS_SERVO_MODE 3

DAC_SERVO_MODE 4

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Mode. Valid range 1...4

Example

SetOutputMode0=1

P<N>=<M>

or

P<N>

Description

Get or Set [PID](#) Proportional Gain.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

Proportional Gain value. The units of the derivative gain are in Output Units/Position Units.

Example

P0=10.0

Pos<N>=<P>

or

Pos<N>

Description

Set or get the measured [position](#) of an axis. Note setting the current position may effect the commutation of any motors based on the position (an adjustment in the [commutation offset](#) may be required).

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<P>

value to be stored into the current position. units are position units. Valid range - any.

Example

Pos0=100.0

ProgFlashImage

Description

Program entire FLASH image, downloaded using [LoadFlash](#) commands, to FLASH Memory.

Parameters

None

Example

ProgFlashImage

PWM<N>=<M>

Description

Set PWM channel to [locked anti-phase mode](#) and to specified value.

See [PWM Description](#) and [Analog Status Screen](#).

Parameters

<N>

PWM channel number. Valid range 0...7

<M>

PWM value. Valid range -255...255.

Example

PWM0=-99

PWMC<N>=<M>

Description

Set PWM channel to Current Mode and to specified value. PWM Channel will operate in closed loop current mode.

See [Analog Status Screen](#).

Parameters

<N>

PWM channel number. Valid range 0...7

<M>

PWM value. Valid range -1000...1000. 1 count = 35 Amps/1024 = 34.2ma

Example

PWM0=-99

PWMR<N>=<M>

Description

Set PWM channel to [recirculate mode](#) and to specified value.

See [PWM Description](#) and [Analog Status Screen](#).

Parameters

<N>

PWM channel number. Valid range 0...7

<M>

PWM value. Valid range -511...511.

Example

PWMR0=-99

ReadBit<N>

Description

Displays whether an actual hardware I/O bit N (0...30) or Virtual IO bit (32...63) is high (1) or low (0) . A bit defined as an output (See [SetBitDirection](#)) may also be read back.

Parameters

<N>

Bit number to read. Valid range - 0...63

Example**ReadBit0**

Reboot!**Description**

Causes complete power up reset and re-boot from flash memory.

Parameters

None

Example**Reboot!**

SetBit<N>**Description**

Sets an actual hardware I/O bit N (0...30) or Virtual IO bit (32...63) to high (1) .

Parameters**<N>**

Bit number to set. Valid range 0...63

Example**SetBit0**

SetBitBuf<N>**Description**

Inserts into the [coordinated move buffer](#) a command to set an I/O bit N(0...30) or Virtual IO bits (32...63) (actual IO bits must be defined as an output, see [SetBitDirection](#))

Parameters**<N>**

Bit number to set. Valid range 0...63

Example

SetBitBuf0

SetBitDirection<N>=<M>

Description

Defines the direction of an I/O bit to be an input or output.

See also [Digital I/O Screen](#).

Parameters

<N>

Bit number to assign. Valid range 0...30

<M>

Direction 0 = input, 1 = output

Example

SetBitDirection0=1

SetGatherDec <N> <M>

Description

Writes a single word to the [Gather Buffer](#) at the specified offset. A single 32-bit value specified as a signed decimal integer number will be stored.

The corresponding value may be accessed by a **KMotion** user program using the pointer : **gather_buffer**. This pointer should be cast as an integer pointer in order to reference values as integers and to use the same index.

See also [GetGatherDec](#), [GetGatherHex](#), [SetGatherHex](#)

Parameters

<N>

Offset into gather buffer, specified as a decimal offset of 32 bit words. Valid range 0...1999999

<M>

Value to be stored. Valid range -2147483648...2147483647

Example

SetGatherDec 1000 32767

SetGatherHex<N> <M>

<H> <H> <H> . . .

Description

Writes a *multiple* words to the [Gather Buffer](#) beginning at the specified offset. 32-bit values specified as a unsigned hexadecimal numbers must follow with 8 words per line separated with spaces. Since this command spans several lines, it may only be used programmatically in conjunction with a [KMotionLock](#) or [WaitToken](#) command so that it is not interrupted.

The corresponding values may be accessed by a **KMotion** user program using the pointer : **gather_buffer**. This pointer should be cast as an integer pointer in order to reference values as integers and to use the same index.

See also [GetGatherDec](#), [GetGatherHex](#), [SetGatherDec](#)

Parameters

<N>

Offset into gather buffer, specified as a decimal offset of 32 bit words. Valid range 0...1999999

<M>

Number of value to be stored, specified as a decimal number. Valid range 0...19999999

<H> <H> <H> . . .

Values to be stored. Specified as unsigned Hexadecimal values. Valid range 0...FFFFFFFF.

Example

SetGatherHex 0 3

FFFFFFFF FFFFFFFF FFFFFFFF

SetPersistDec <O> <D>

Description

Write a single word into the Persistent *UserData* Array. Persistent *UserData* Array is a general purpose array of 100 32-bit words that may be used as commands, parameters, or flags between any host applications or **KMotion** user programs. The array resides in a *persistent* memory area, so that if a value is set as a parameter and the User Programs are *flushed*, the value will persist permanently.

The corresponding value may be accessed by a **KMotion** user program as the integer variable : `persist.UserData[offset]`.

See also [GetPersistDec](#), [GetPersistHex](#), [SetPersistHex](#)

Parameters

<O>

Offset into the user data array specified as a decimal number. Valid Range 0 ... 99.

<D>

Value to be written to the array. Specified a signed decimal number. Valid Range -2147483648 ... 2147483647

Example

SetPersistDec 10 32767

SetPersistHex <O> <H>

Description

Write a single word into the Persistent *UserData* Array. Persistent *UserData* Array is a general purpose array of 100 32-bit words that may be used as commands, parameters, or flags between any host applications or **KMotion** user programs. The array resides in a *persistent* memory area, so that if a value is set as a parameter and the User Programs are *flushed*, the value will persist permanently.

The corresponding value may be accessed by a **KMotion** user program as the integer variable : `persist.UserData[offset]`.

See also [GetPersistDec](#), [GetPersistHex](#), [SetPersistDec](#).

Parameters

<O>

Offset into the user data array specified as a decimal number. Valid range 0 ... 99.

<H>

Value to be written to the array. Specified an unsigned hexadecimal number. Valid range 0...FFFFFFFF

Example

SetPersistHex 10 FFFFFFFF

SetStartupThread<N> <M>

Description

Defines whether a user thread is to be launched on power up.

Parameters

<N>

Selected User Thread. Valid range 1...7

<M>

Mode : 1=start on boot, 0=do not start on boot.

Example

SetStartupThread0 1

SetStateBit<N>=<M>

Description

Sets the state of an actual hardware I/O bit N (0...30) or Virtual IO bit (32...63) to either low (0) or high (1) . Actual I/O bits must be defined as an output, see [SetBitDirection](#).

Parameters**<N>**

Bit number to set. Valid range 0...63

<M>

State. Valid range 0...1

Example**SetStateBit0=1**

SetStateBitBuf<N>=<M>**Description**

Inserts into the [coordinated move buffer](#) a command to set the state of an I/O bit N(0...30) or Virtual IO bits (32...63) (actual IO bits must be defined as an output, see [SetBitDirection](#))

Parameters**<N>**

Bit number to set. Valid range 0...63

<M>

State. Valid range 0...1

Example**SetBitBuf0****SetStateBitBuf0=1**

SlaveGain<N>=<S>

or

SlaveGain<N>**Description**

Sets or gets the Slave Gain for the axis. See also [MasterAxis](#) for more information

Parameters

<N>

Selected Axis for command. Valid range 0 ... 7.

<S>

Slave Gain. Any floating point value positive or negative.

Example

SlaveGain0=-1.0

or

SlaveGain0

StepperAmplitude<N>=<M>

or

StepperAmplitude<N>

Description

Set or get the nominal output magnitude used for axis if in MicroStepping [Output Mode](#) to the specified value. This will be the output amplitude when stopped or moving slowly. If [Lead Compensation](#) is used, the amplitude while moving may be higher.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<M>

PWM Stepper Amplitude. Valid range 0...255

Example

StepperAmplitude0=250

TrigThread <S>

Description

Triggers a coordinated motion threading operation. The coordinated motion path in the coordinated motion buffer begins execution synchronized with the Spindle motion. The Speed specified will be used as the baseline speed such that if the actual spindle speed is equal to the base speed, then Pseudo Time will progress the same as real time. Otherwise Pseudo time will be adjusted to match the spindle motion

See also: [ConfigSpindle](#) and [GetSpindleRPS](#)

Parameters

<S>

Base Spindle Speed in revs per second. Range: Any floating point value.

Example

TrigThread 10.0

Vel<N>=<V>

or

Vel <N>

Description

Get or Set the max velocity for independent moves.

Parameters

<N>

Selected Axis for command. Valid range 0...7.

<V>

The max velocity. Units are in *Position units* per sec

Example

Vel0=100.0

Version**Description**

Display DSP Firmware Version and Build date in the form:.

KMotion 2.22 Build 22:26:57 Feb 16 2005

Note it is important that when C Programs are compiled and linked, they are linked to a firmware file, DSP_KMotion.out, that matches the firmware in the KMotion where they will execute.

Parameters

None

Example**Version**

Zero<N>**Description**

Clear the measured position of axis. Note for an axis that uses the [Position](#) to perform brushless motor commutation, the [commutation offset](#) may be required to be adjusted whenever the position measurement is changed.

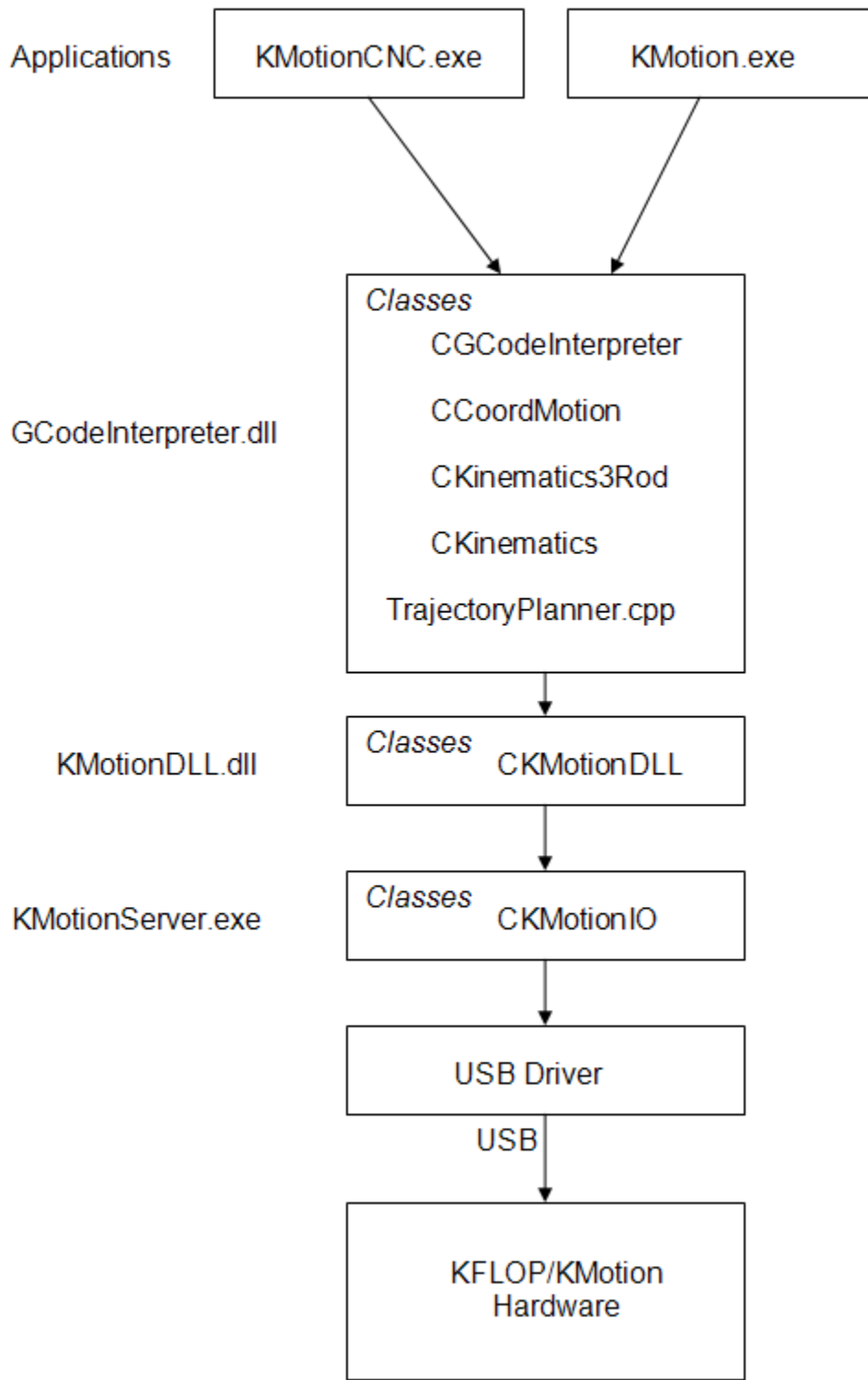
Parameters

<N>

Selected Axis for command. Valid range 0...7.

Example

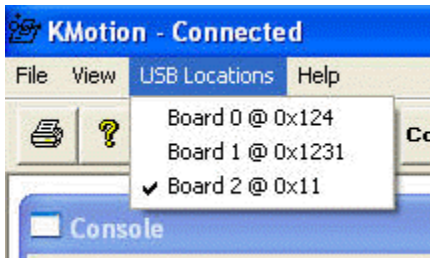
Zero0

KFLOP/KMotion Interface Libraries

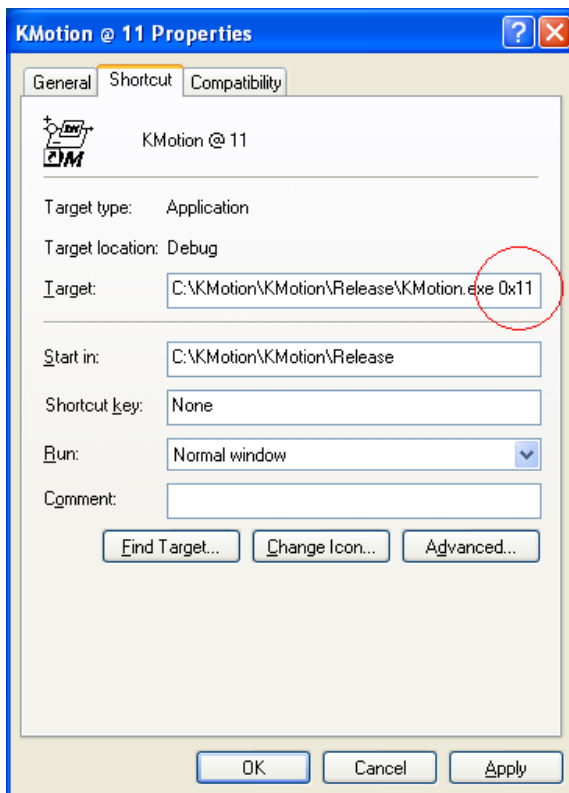
Using Multiple KMotion Boards

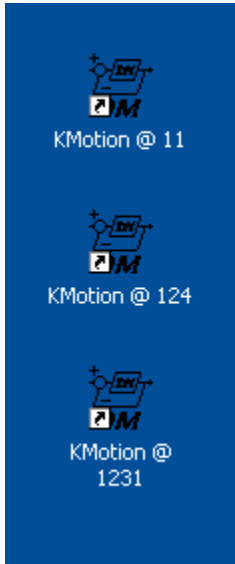
The **KMotion** Driver Library allows multiple PC processes (applications), each running multiple threads of execution, to communicate with multiple **KMotion** boards simultaneously. Each **KMotion** board is identified by a USB *location identifier* where it is connected. A USB location identifier is a 32 bit integer. USB devices are arranged in a tree structure of hubs and nodes. Each hexadecimal digit of the USB location specifies a branch in the tree structure. For the purposes of the **KMotion** Driver Library, a USB location identifier may simply be considered a unique integer that will remain the same as long as the structure of the USB tree is not altered. Adding or removing USB devices will not change the location of a connected **KMotion** board.

Selecting the USB Locations menu of the **KMotion** Setup and Tuning application, will display a list of all currently connected **KMotion** boards. The Checkmark indicates which board the application is currently communicating with. To switch to a different board, select the USB location from the list.



When launching the **KMotion** Setup and Tuning application, a command line parameter may be specified to connect to a specific USB location (see below on how to setup a shortcut to connect to a specific location). Multiple shortcuts may be setup to connect to individual boards.



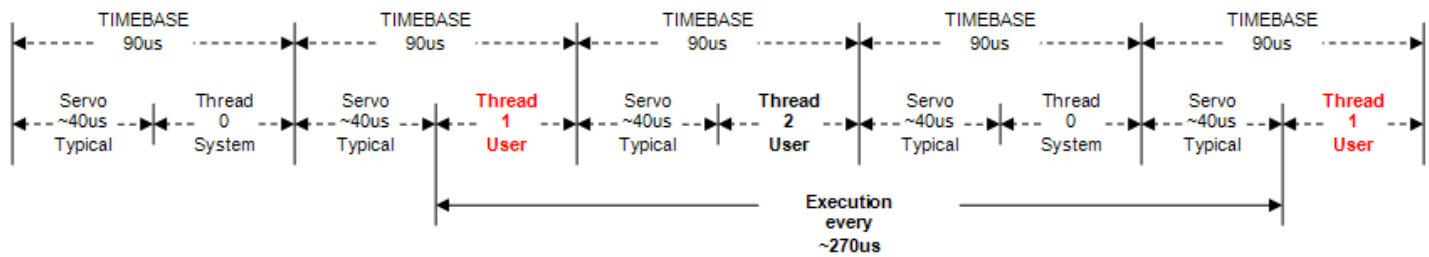


The **KMotion** Driver Library has a function to list the USB locations of all currently connected **KMotion** boards. See:

`int ListLocations(int *nlocations, int *list);`

When making **KMotion** Driver Library function calls specify the USB *location identifier* of the desired board as the board parameter shown in the example below. Specifying a board value of 0 may be used if there is only one board in a particular system. This will result in a connection to the first available board.

```
int CKMotionDLL::WriteLineReadLine(int board, const char *s, char  
*response)
```

KMotion/KFlop Preemptive Multitasking

KMotion/KFlop uses a simple method of preemptive multitasking. User Programs (Threads) and the System Thread context switch every Servo Interrupt and sequence in a round robin order.

The above example shows a case where two User Threads are currently active.

The C function `WaitNextTimeSlice()` can be used to wait until the next context switch occurs and return immediately at the beginning of the next time slice. This can assure that the User Program can execute for a few microseconds without being interrupted and at a very stable rate (with a few microseconds of jitter).

The time period between executions of each user thread is a function of the number of active User Threads:

$$\text{Period} = (\# \text{ User Threads} + 1) * \text{TIMEBASE}$$

KFLOP - RS232

KFLOP contains a UART that can allow KFLOP User C Programs to perform serial communication with other 3rd party devices.

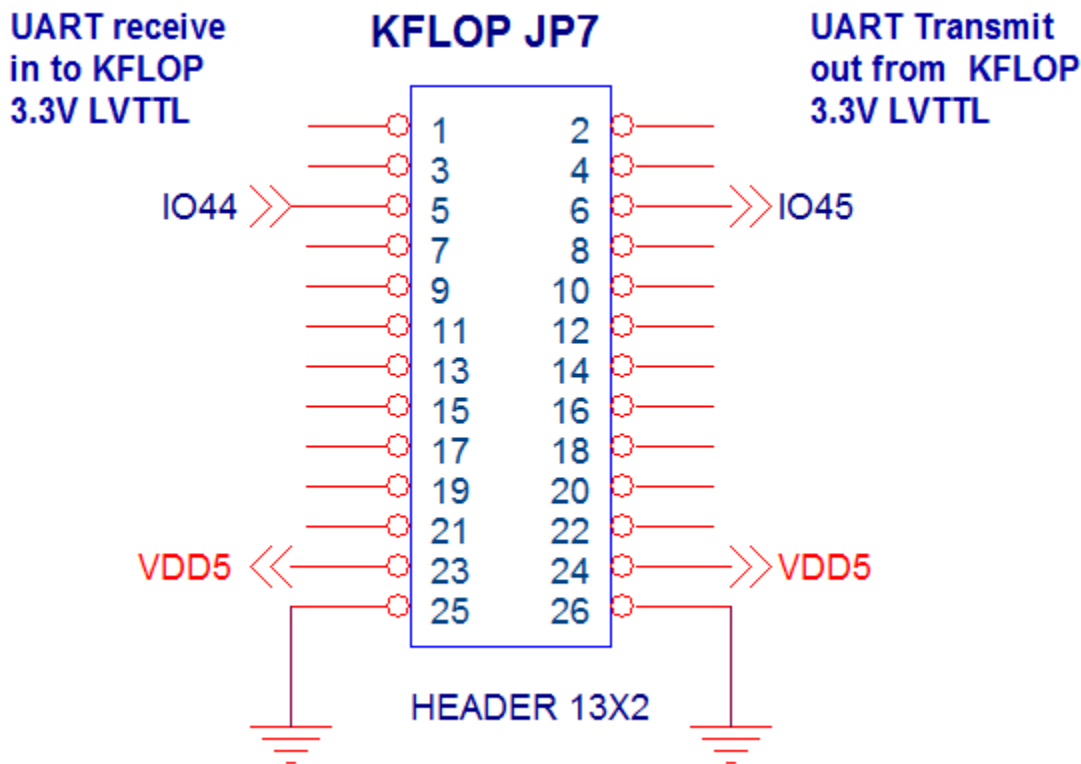
KFLOP itself contains the serial communication UART but does not have the circuitry to drive or receive the signals at the +3 to 25V to -3 to -25V voltages levels specified by the RS232 standard. The transmit and receive signals to/from KFLOP are 3.3V LVTTTL logic signals. Where a low logic level (<0.4V) represents the RS232 Space Level (>+3V) and a high logic level represents the RS232 Mark Level (< -3V).

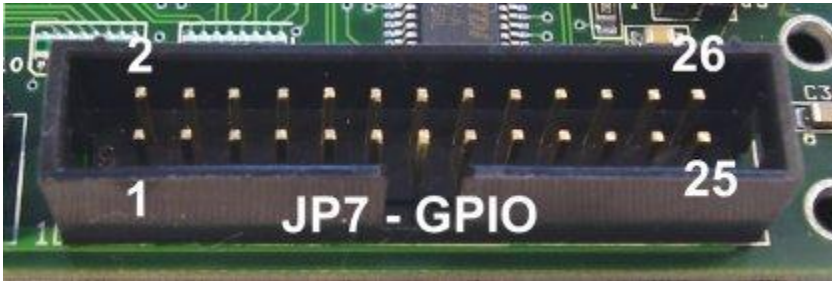
Note that the signals that the signals coming directly from KFLOP are not RS232 compatible. Connecting KFLOP inputs directly to RS232 is likely to cause damage.

Some serial devices may be compatible with 3.3V logic. Also many 3rd party converters are available. A Internet search found this one (we haven't tested it).

<http://www.commfront.com/TTL-RS232-RS485-Serial-Converters/RS232-TTL3.3V-Converter.htm>

Our Kanalog board has a LVTTTL to converter on board (see next section below).

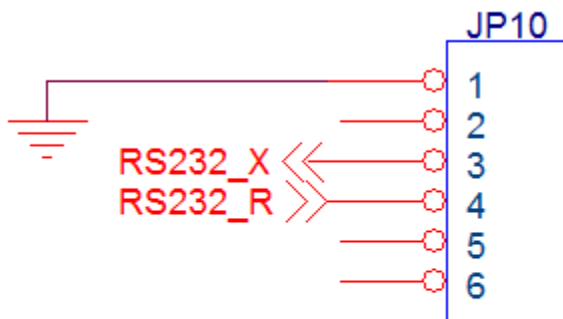




KFLOP + Kanalog - RS232

Kanalog contains circuitry to convert the KFLOP UART logic levels to standard RS232 voltage levels. The RS232 signals are accessible from JP10 which is a standard RJ12 phone jack.

The pinout is shown below. The pinout is designed to be 1:1 compatible with certain PLC devices. Note that a phone cable with all 6 wires populated is required in order to make the pin 1 ground connection.



RJ12 TYCO 5555165-1

KFLOP UART Software

The KFLOP FPGA implements a programmable baud rate UART with double buffering of 1 character on transmit and receive. Currently a KFLOP User C program must be used to make use of the UART. It is up to the User to program whatever is necessary to control any specific device.

KFlop User Programs with 1 active thread execute every 180us. (See [here](#) for more info). This allows rates up to 38400 baud without loss if a character is read every time slice (10 bits at 38400Hz = 260us). Data transmitted and received is always 8 bits. If parity is required it should be handled by the User C program.

To set the baud rate write a defined 8-bit baud rate divisor to an FPGA Register (RS232_BAUD_REG)

To transmit a character an 8-bit character is written to an FPGA register (RS232_DATA).

To receive a character read an 8-bit value from an FPGA register (RS232_DATA)

A status register (RS232_STATUS) provides 2 bits of status that can be used to determine if a character has been received (RS232_DATA_READY) and if it is possible to transmit a character (RS232_TRANSMIT_FULL)

The following definitions have been added to the KMotionDef.h file.

```
//RS232 FPGA Register Definitions

#define RS232_STATUS 0xc1 // Status Reg Address
#define RS232_DATA 0xc0 // 8 bit data read/write reg address
#define RS232_DATA_READY 0x01 // Data ready to read status mask
#define RS232_TRANSMIT_FULL 0x02// Transmit buffer full status mask

#define RS232_BAUD_REG 0xc1 // Set Baud rate 8-bit divisor Reg Address
#define RS232_BAUD_115200 ((16666666/115200/16)-1)// 8-bit divisor value
to set 115200 baud
#define RS232_BAUD_57600 ((16666666/57600/16)-1) // 8-bit divisor value
to set 57600 baud
#define RS232_BAUD_38400 ((16666666/38400/16)-1) // 8-bit divisor value
to set 38400 baud
#define RS232_BAUD_19200 ((16666666/19200/16)-1) // 8-bit divisor value
to set 19200 baud
#define RS232_BAUD_9600 ((16666666/9600/16)-1) // 8-bit divisor value to
set 9600 baud
#define RS232_BAUD_4800 ((16666666/4800/16)-1) // 8-bit divisor value to
set 4800 baud
```

Note if KFLOP is to be used without Kanalog the UART IO pins must be activated by executing the following line of code one time:

```
FPGA(KAN_TRIG_REG)=1;
```

Transmit Example (RS232Send.c example)

```
#include "KMotionDef.h"

void SendChar(char c)
{
while (FPGA(RS232_STATUS) & RS232_TRANSMIT_FULL) ;
FPGA(RS232_DATA) = c;
}

main()
{
```

```
int i;
SetBitDirection(45,1);
FPGA(RS232_BAUD_REG) = RS232_BAUD_38400;
// FPGA(KAN_TRIG_REG) = 1; // enable Kanalog to get RS232 working
for (i=0;i<100;i++)
{
    SendChar('A');
}
}
```

Receive Example (RS232Read.c example)

```
#include "KMotionDef.h"

void ReceiveChar()
{
    // wait for data in buffer
    while ((FPGA(RS232_STATUS) & 1)==0);
    return FPGA(RS232_DATA);
}

main()
{
    SetBitDirection(45,1);
    FPGA(RS232_BAUD_REG) = RS232_BAUD_38400;
    // FPGA(KAN_TRIG_REG) = 1; // enable Kanalog to get RS232 working
    for (;;)
    {
        while ((FPGA(RS232_STATUS) & RS232_DATA_READY) == 0) ;
        printf("%X\n",ReceiveChar());
    }
}
```

G Code Quick Reference

G Codes

G0 X3.5 Y5.0 Z1.0
A2.0 (Rapid move)

8

G90 Absolute
Coordinates

G91 Relative
Coordinates

G92 Set Global Offset
G92 X0Y0Z0

G92.1 Clear Global
Offset

M Codes:

M0 (Program Stop)

M2 (Program End)

M3 Spindle CW

M4 Spindle CCW

M5 Spindle Stop

M6 Tool Change

M7 Mist On

M8 Flood On

M9 Mist/Flood Off

M98 Pxxx Call

Subroutine

M99 Return from Sub

Other Codes:

F (Set Feed rate
in/min or mm/min)

S (Spindle Speed)

D (Tool)

O Subroutine Label

Comments:

(Simple Comment)

(MSG,OK

toContinue?)

(CMD,EnableAxis0)

(BUF,SetBitBuf29)

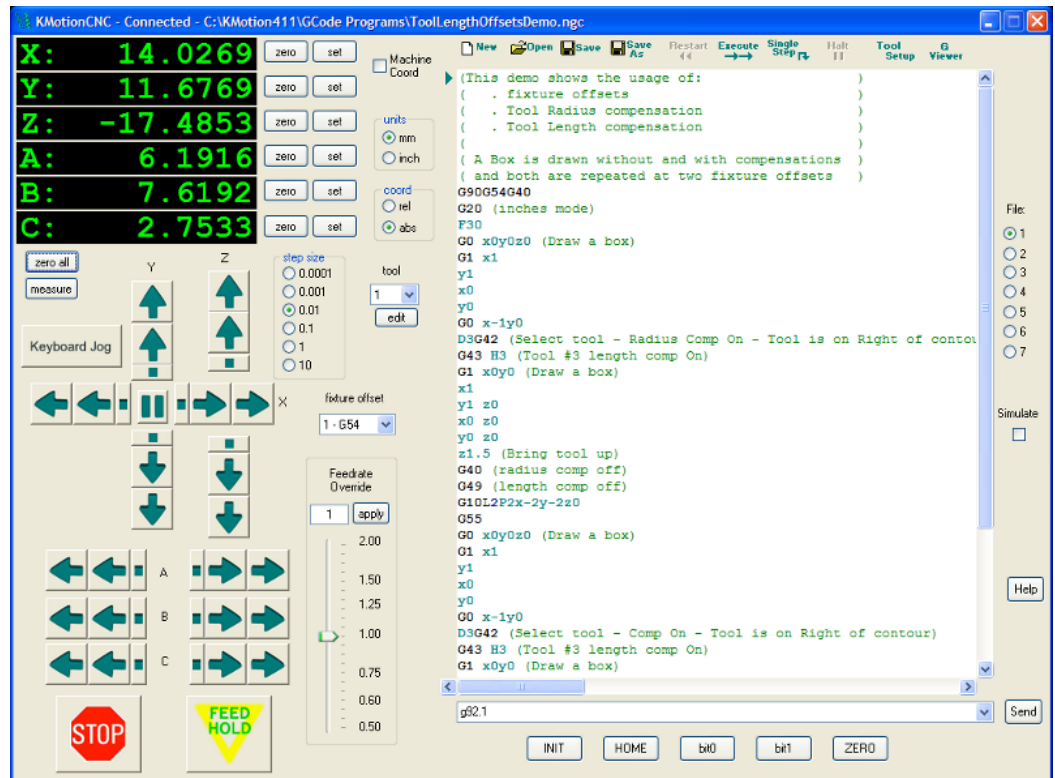


Other KMotionCNC Screens

[G Code Viewer Screen](#)

[G Code Viewer Setup Screen](#)

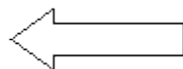
[Tool Setup Screen](#)



(Click on Image to Jump to related help)

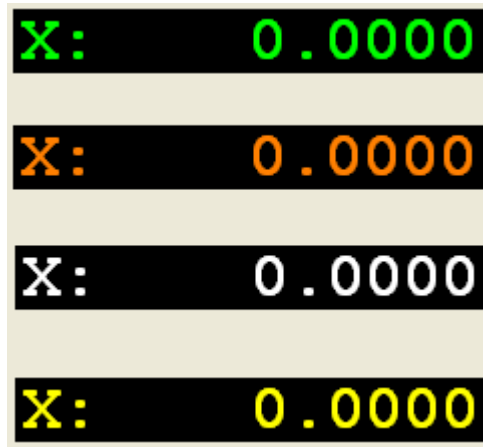
KMotionCNC allows the user to edit, execute, and view G Code Programs. GCode is a historical language for defining Linear/Circular/Helical Interpolated Motions often used to program numerically controlled machines (CNC Machines).

KMotion has various screen "faces". The one shown above is for "Basic 6 axes". Others may be selected on the [Tool Setup Screen](#).



See the Quick Reference at left for commonly used G Code commands.

Display



The 4 axis Display along the top of the screen indicated the current position of each axis. The units of the display are in either mm or inches depending on the current mode of the interpreter (see [Coordinate System Units](#)).

The displayed position will match the g -code programmed position (i.e. last G1 commanded position) which is not necessarily the actual machine tool position if global or fixture offsets are in use.

The color of the display gives an indication of current status.

Green - indicates normal status, hardware is connected, axis is enabled, and the displayed position is the current tool position in GCode coordinates.

Orange - indicates normal status, hardware is connected, axis is enabled, and the displayed position is the current tool position in Raw Machine Coordinates (G53 without any Global Offset (G92) or Fixture Offset (G54+).

White - indicates simulation mode is selected. The displayed position is the current position after the last line of interpreted G code.

Yellow - indicates hardware disconnected or axis disabled. The displayed position is invalid

Zero/Set Origin



Allows zeroing or setting the respective axis. This is accomplished by

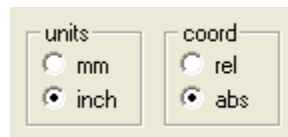
adjusting the Global Offset (G92). The original Raw Machine Coordinates will remain unchanged, so the effect of these operations will only be apparent if displaying Machine Coordinates is de-selected, see below. For further information see [system offsets](#).

Coordinate System Origin

☒ Machine Coord When checked, the displayed position is the current tool position in Raw Machine Coordinates (G53 without any Global offset (G92) or Fixture Offset (G54+). Machine coordinates are relative to the fixed machine home position.

Unchecked displays the normal GCode Coordinate relative to the "floating" origin which may be moved by changing either the global offset (G92) and/or a Fixture Offset.

Coordinate System Units / Mode



Displays the current mode of the G code interpreter.

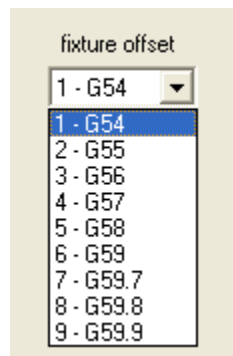
G20 selects English Inch units

G21 selects Metric mm units

G90 selects Absolute Coordinates

G91 selects Relative Coordinates

Fixture Offset

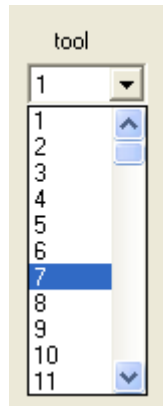


Displays and allows changing of the current Fixture Offset. *KMotionCNC* supports 9 Fixture offsets. Each Fixture may be programmed to introduce an arbitrary x,y,z,a offset. Use the **G10L2Pn** command to set the offset associated with the fixture #n. An example might be:

G10 L2 P3 X10.0 Y20.0 Z30.0 which sets Fixture Offset #3 to (10,20,30)

Executing the command **G56** (or by selecting 3 - G56 in the drop down list) will cause Fixture offset #3 to be in use in subsequent commands until a different Fixture is selected. (See also - [G Code Offsets](#)).

Tool



Displays and allows changing of the currently selected Tool. *KMotionCNC* supports up to 99 tool definitions. GCode selects tools using several different commands for different purposes. The value displayed here is the value selected by the **T** command, which is the tool to be loaded into the spindle by a automatic tool changer. The T command is normally followed by a M6 command that receives the tool number and physically loads the tool. Selecting a tool in the the dropdown list will effectively cause a T#M6 command to be executed, where # is the selected tool number.

Besides changing tools with the T command, D and H commands are used to apply tool properties from the tool table. The tool definitions are specified in a text file that is selected on the [ToolSetup Screen](#).

A tool definition consists of the tool number (FMS), the pocket where it is located if there is a tool changer available (POC), the length of the tool (LEN), the diameter of the tool (DIAM), and an optional comment.

Executing the **D#** command will select which Tool parameters are to be used for radius compensation (**G40,G41,G42**).

Executing the **H#** command will select which Tool parameters are to be used for length compensation (**G43,G49**).

See below for an example Tool Table

```
POC FMS LEN DIAM COMMENT
1 1 0.0 0.0 first tool
2 2 0.0 0.0
3 3 1.0 0.5
4 4 2.0 1.0
32 32 0.0 0.0 last tool
```

The Tool Table may also be edited in a window using the **edit** button.

File New / Open/ Save



This group of pushbuttons allow a G Code file to be loaded or saved to or from the edit window. The edit window allows the user to quickly switch between 7 loaded G Code files. Once a file is loaded into one of the edit windows, the name of that file will persist between sessions.

Execute Controls



This group of pushbuttons allow the control of G Code execution. **Restart** will reset the instruction pointer to the first line of the file. **Execute** will begin continuous execution from where the current instruction pointer is located on the left of the edit window. **Single Step** will execute one single line of G code where the current instruction pointer is currently pointing. Note that the instruction pointer may be moved to any line by right clicking on the line within the edit window and selecting Set Next Statement.

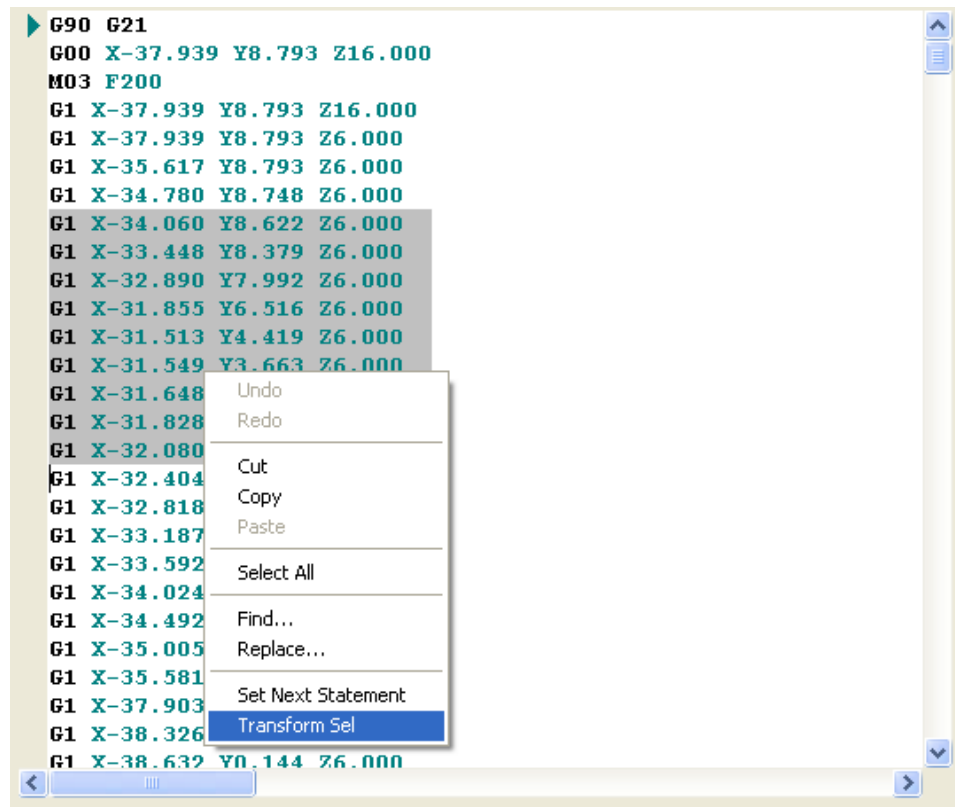
Halt will cause a feedhold to be initiated so that any coordinated tool motion will decelerate along the intended path. Once motion comes to a complete stop, the GCode Interpreter will abort and back up to the state of the line of GCode that created the current Tool motion.

Show Tool Setup / G Code Viewer Screens

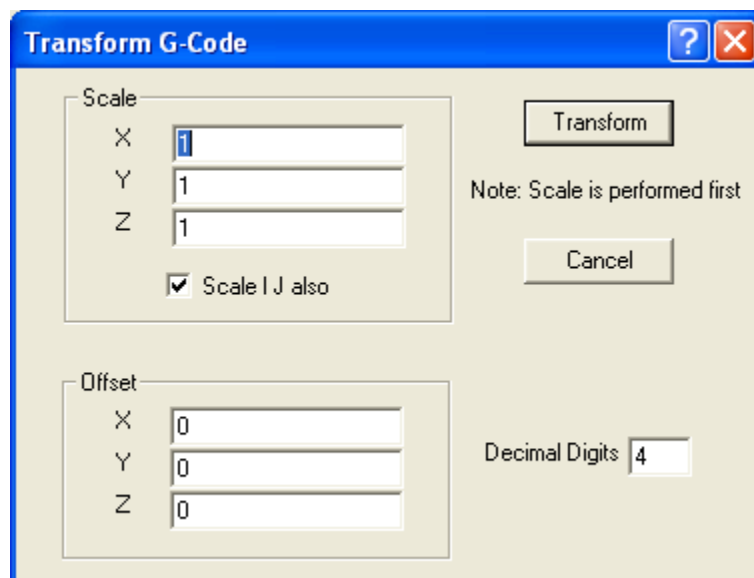


This group of buttons bring additional screens into view. The [Tool Setup Screen](#) is used to configure the system's parameters. Machine Axis distance/velocity/accelerations. M Code and User Button Actions, Tool and Setup definition files, and Jog Button and Joystick rates. The [G Code Viewer Screen](#) allows real-time, 3D viewing of the machine tool paths either during actual machine operation or during simulation.

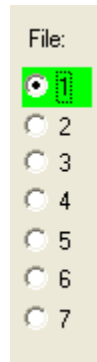
G Code Edit Window



The Edit Window displays the loaded G-Code and allows editing. G-Code color syntax highlighting makes the code more readable. A right mouse click will bring up the context menu shown. Note that Transform Sel will bring up a utility dialog (shown below) that allows the selected G-Code to be scaled or offset.

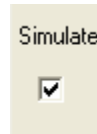


File Selector



The file selector shows which of the 7 loaded G Code files is currently active for editing. The Main Window Title also displays the loaded filename for the selected file. The file number is highlighted in green when that file is currently executing G Code. Only one G Code file is allowed to execute at a particular time.

Simulate



Enables Simulation Mode which allows viewing and verification of a G Code Program with or without any actual hardware connected. When Simulation mode is enabled no actual machine motion will be made. Executing or Single Stepping through a G Code program will change the Displayed Position and Plot the machine tool path on the [G Code Viewer Screen](#). In Simulation Mode the Numeric [Display](#) Color changes to white to indicate the display is not showing the actual machine tool position. While in Simulation mode the [Jog Buttons and Gamepad buttons](#) will also change the displayed position and tool position on the G Code Viewer Screen without causing any actual machine tool motion.

Feed Hold



Feed Hold may be used to immediately decelerate all axes to a controlled stop. Any coordinated motion in progress will decelerate along the defined buffered path. When stopped in feed hold, the button will be shown in a toggled state. To resume motion toggle the feed hold button off. Note that while stopped that the current GCode line may be advanced one or more lines past the current tool position because the Interpreter and Trajectory planner works ahead to allow motion to be buffered. [Halt](#) may also be selected to exit the feedhold state. Halt will cause the Interpreter to abort and back up to the GCode line that generated the current tool position.

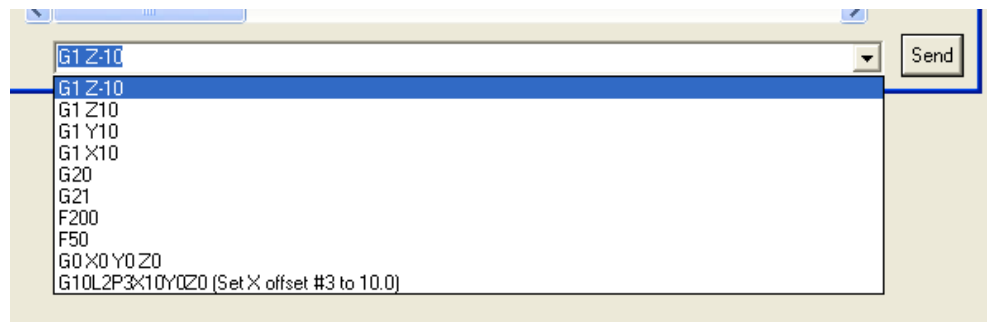
Emergency Stop



Emergency Stop may be used to immediately stop all motion. Any commands in motion will be aborted and all axes will be disabled. After depressing Emergency Stop the system must be re-initialized and the G Code Interpreter state will be lost. Use Halt to stop in a controlled manner after the next line of GCode has been completed.

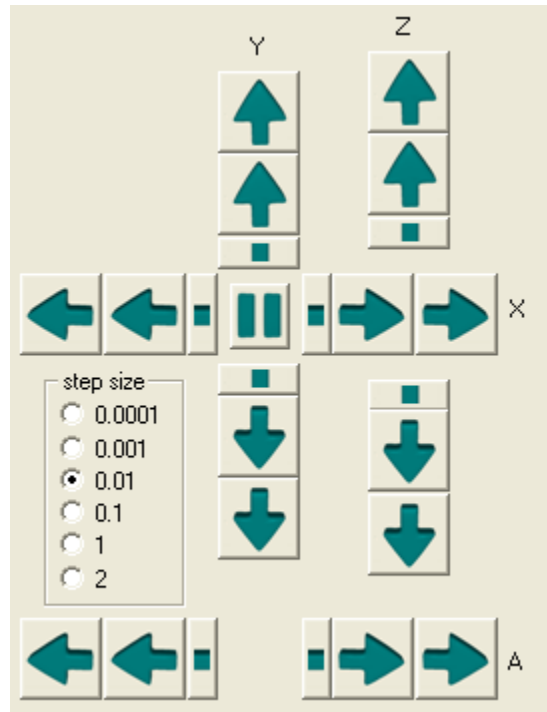
The ESC key may also be use to initiate an Emergency Stop whenever the KMotionCNC Screen has the focus.

Manual Entry



The Manual Entry cell allows the user to quickly enter a single line of G Code and Send it to the interpreter for execution. The last 10 entered commands are saved in a drop down list for quick re-entry.

Jog Buttons



The Jog buttons may be used to move any of the axes. Pushing and holding any of the arrow buttons will cause continuous motion. There are 2 buttons in each direction for each axis. The second button moves at twice the rate as the first. The speeds for each axis may be specified in the [Tool Setup Screen](#).

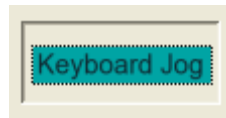
There is also a Step button (square dot) for each axis and direction that will move an exact step size (as specified in the step size selection). The center button will stop a step in progress (useful for aborting large steps).

A USB Gamepad such as the one shown below may also be used to Jog the System. Simply connect the Gamepad and it should become immediately active. The left Joystick controls x and y and the right joystick controls z and a. The same speed parameters in the Tool Setup Screen control both the Jog pushbuttons on the screen and the Gamepad.

The Jog buttons and Gamepad are also active in simulation mode

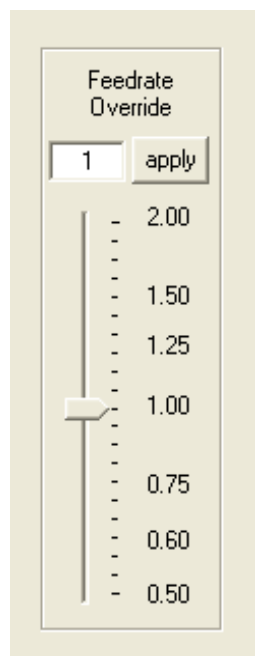


Keyboard Jog



Toggling on Keyboard Jog allows keyboard keys that are normally used for Windows navigation to be used for Jog Functions. These include keys such as arrow, page up/down, +, - etc. Hovering the mouse cursor over a Button will display a tool tip that indicates a keyboard key that may be used instead of the mouse. Mouse clicking into an edit cell that requires the use of arrow keys will automatically de select Keyboard Jogging. The F2 key may be used to toggle Keyboard Jogging.

Feed Rate Override



Feed Rate Override provides a means to adjust the feedrate while the machine is in operation without having to modify the G Code. The Feed Rate is specified within the G Code using the F command and the Feed Rate Override is a multiplicative factor that is applied to that value. For example F100 would specify a Feed rate of 100 inches/minute (or 100 mm/minute if the interpreter is in metric mm mode), with a feed rate override of 1.5 the actual attempted feed rate would be 150 inches/minute (or 150 mm/minute in mm mode).

Note that this speed will only be used if all the axes involved will remain below the maximum allowed speeds set on the [Tool Setup Screen](#). Additionally, short vectors with sharp corners (greater than the specified break angle) that require stopping may be limited due to acceleration limits. KMotionCNC uses complex algorithms to ensure that velocities and accelerations remain below user specified limits. So if the Feed Rate Override or (Specified Feed rate itself) doesn't seem to be having the expected result, check if the maximum velocities and accelerations are limiting.

The Feed Rate Override may be changed either by using the slider or by typing a value and pressing the apply button.

Note that because motions are planned ahead and downloaded to the Motion Controller ahead of time, that the feed rate override will take a short amount of time to have an effect. The amount of time that the trajectory planner looks ahead is specified on the Tool Setup Screen and is normally set at from 1 to 3 seconds. The main limitation to making this value very short is the worst case response time of Microsoft Windows™ and the PC hardware being used.



Custom Buttons

KMotionCNC allows up to 5 Custom Buttons to be displayed and defined for special operations. Which of these buttons are visible, what they display as a title, and what action they perform are all definable on the [Tool Setup Screen](#).

The actions that the buttons perform are defined by the User in the same manner as the actions that M Codes perform. These may be simple actions such as setting an Output to turn something on or may be a complex operation that involves invoking a program. Normally one or more buttons will be used to initialize and configure the motion controller and/or home the machine.

Other GCode Commands

KMotion's G Code interpreter was derived from the Open Source EMC G Code Interpreter. Click here for the [EMC User Manual](#) (Only the G Code portions of the manual, Chapters 10-14 pertain to **KMotion** G Code)

Specially coded comments embedded within a GCode program may be used to issue **KMotion** Console Script commands directly to **KMotion**.

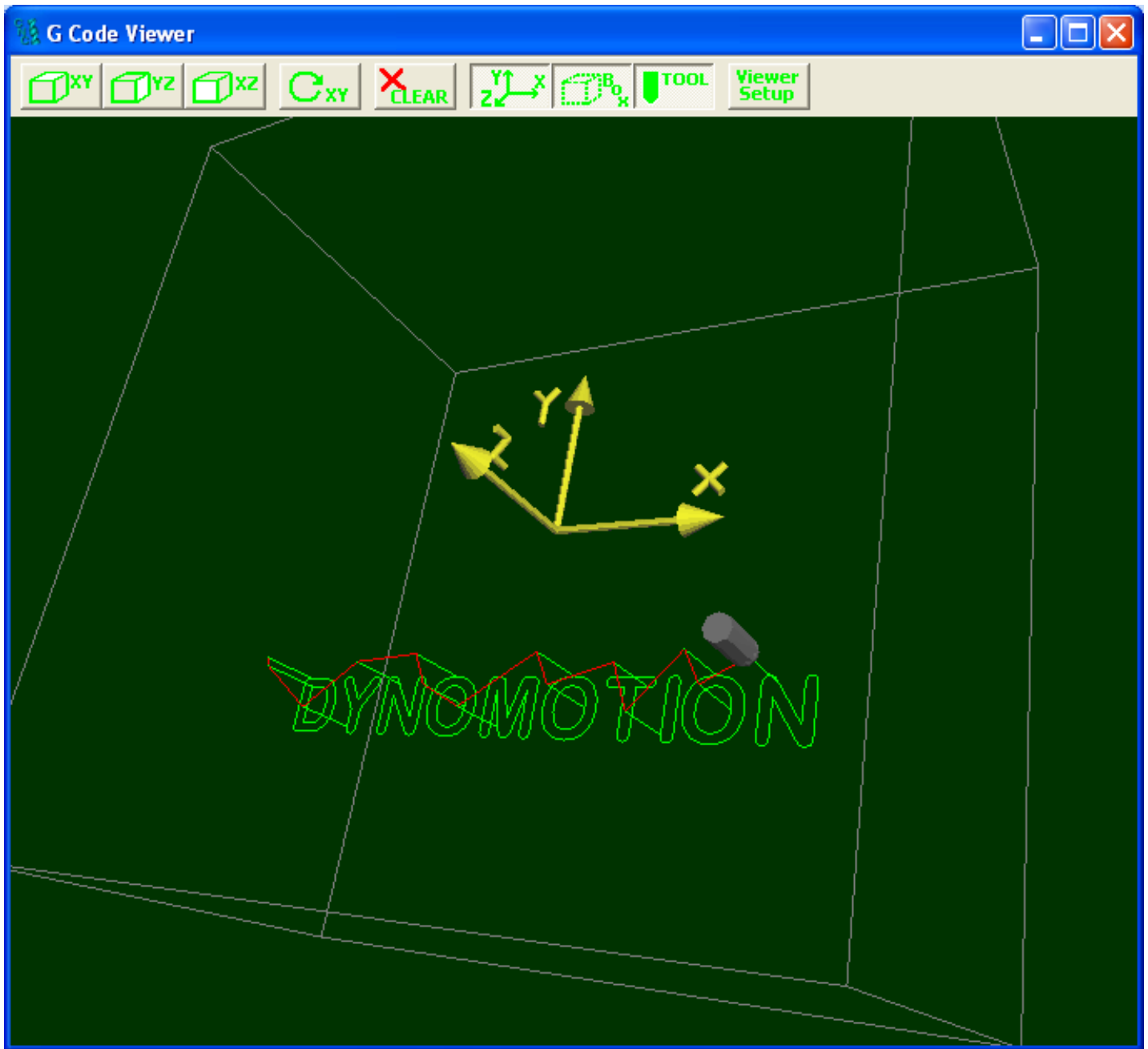
A comment in the form: (CMD,xxxxxx) will issue the command xxxxxx immediately to **KMotion** as soon as it is encountered by the Interpreter. Any **KMotion** command that does not generate a response may be used in this manner.

A comment in the form: (BUF,xxxxxx) will place the command xxxxxx into **KMotion's** coordinated motion buffer. Coordinated motion sequences are download to a motion buffer within **KMotion** before they are executed. This guarantees smooth uninterrupted motion. The BUF command form allows a command to be inserted within the buffer so they are executed at an exact time within the motion sequence. Only the following **KMotion** Script commands may be used in this manner.

SetBitBuf, ClearBitBuf, SetStateBitBuf.

Additionally, a comment in the form: (MSG,xxxxxx) will pause GCode Execution and display a pop-up message window displaying the message xxxxxxxx.

G Code Viewer Screen



The G Code Viewer Screen displays the 3D motions of the machine tool as a GCode program is executing. To Display an entire G Code Program quickly without any physical motion of the machine tool, select [Simulate](#) on the main [KMotionCNC Screen](#) and execute the program. Linear and Circular Feed Motions (G1, G2, G3) are displayed as green paths. Rapid Motions (G0) are displayed as red lines. Note that rapid motions may not actually be performed as straight lines since during rapid motions each axis moves independently as quickly as possible.

Click and drag the **left** mouse Button to translate **up/down/left/right**.

Click and drag the **right** mouse Button to translate **closer or farther**.

Click and drag **both** mouse Buttons to **rotate**.

Top/Side/Front Views



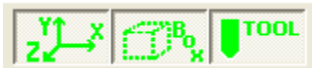
Use these buttons to view directly from the Top, Side, or Front respectively. The camera is positioned at a distance away to view the entire box extents or current path extents whichever is greatest.



This is a latching toggle button. When latched down rotation is in the x y plane about the z axis. When unlatched, rotation is up/down/left/right.



Clears all path vectors. As a G Code Program executes motion paths are saved and displayed. Pushing this button clears all vectors from the internal buffer. All path vectors are also automatically cleared when the first line of a G Code program executes.



These are three latching toggle buttons which determine whether the Axis, Box, or Tool Shapes respectively are to be displayed. When latched down the item is displayed. When latched up the item is hidden. The size and shape of these items may be changed on the [G Viewer Setup Screen](#).



This button brings up the [G Viewer Setup Screen](#) which allows some customization of the G Code Viewer Screen.

G Viewer Setup Screen

G Viewer Setup

Tool Shape

VRML File

Tool Scale

Offset

X	Y	Z
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0.5"/>

Box

Size

X	Y	Z	
<input type="text" value="5"/>	<input type="text" value="5"/>	<input type="text" value="3"/>	Inches

Offset

X	Y	Z	
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	Inches

Axis Scale

Cancel OK

The G Viewer Setup Screen sets the imaging parameters for the [G Code Viewer Screen](#).

Besides the G code tools paths, the G Code Viewer Screen displays several objects, namely a Tool, a Box, and an Axis Symbol. The Tool Object and Axis Symbol are 3D VRML files that may be changed if desired. KMotionCNC comes with default files shown below.

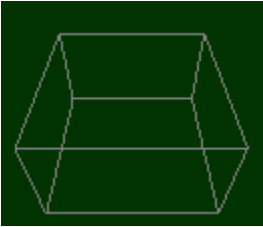
Tool Shape

The default Tool Shape is located at: **<Install Dir>\KMotion\Data\Tool.wrl**, but maybe easily changed by entering or browsing to a new file. An excellent free program to create VRML files is available at <http://www.artofillusion.org>. The default Tool file is a 1 inch diameter sphere at the end of a 1 inch diameter cylinder which is 3 inches long. In the VRML file the origin (0,0,0) is at the center of the sphere. The Tool Scale and offsets allow the Tool Shape to be shifted and scaled as desired. As shown above an offset of (0,0,0.5) shifts the tool such that the origin is at the very tip of the tool. The scale of 0.15 then reduces the tool "size" from 1 inch to 0.15 inch.

Axis Scale



The Axis file name and location is hard coded as: **<Install Dir>\KMotion\Data\Axis.wrl**. The Axis shape is always drawn at the origin. It's size may be changed by changing the Axis Scale value. If a different Axis Shape is desired the Axis.wrl file must be overwritten with a new file.



Box

The Box serves two purposes. If enabled for display, it aids the 3D visualization and perspective of the tool paths. It is also expected to represent the working extents of the machine. When "zooming out" to one of the preset Top, Side, or Front Views, it is used to frame the view extents. Therefore the Box Size parameters should be set to the working extents of the machine. With an offset of 0,0,0 the origin will be located at the center of the Box. This is the preferred arrangement, however if required the Box may be displayed offset by entering Box Offset values other than zero

Tool Setup Screen - M3 - M9

Tool Setup Screen

M3 - M9 M100-M119 User Buttons Tool/Setup Files Trajectory Planner

M3 I/O Bit Set bit 144 to 1

M4 I/O Bit Set bit 145 to 1

M5 Two I/O Bits Set bit 144 to 0 & bit 145 to 0

M6 Execute Prog Thread 2 VAR 9 C File C:\KMotionSrc\C Programs\ToolChange.c >

M7 I/O Bit Set bit 22 to 1

M8 I/O Bit Set bit 23 to 1

M9 None

S DAC Set DAC 0 scale 1 offset 0 min 0 max 1024

Cancel OK Help

(Click on Image to Jump to related help)

The Tool Setup Screen allows *KMotionCNC* to be configured for a particular machine tool. Each machine tool is likely to have different motion resolution, speeds, and acceleration limits. It is also likely to have different I/O requirements with regard to Spindle control and such. Additionally a machine may have different initialization and homing requirements. *KMotionCNC* has a flexible mechanism for defining what type of action is to be performed for various M Codes and Custom Buttons.

G Code Actions - M3 - M9

M3	I/O Bit	Set bit	144	to	1	
M4	I/O Bit	Set bit	145	to	1	
M5	Two I/O Bits	Set bit	144	to	0	& bit 145 to 0
M6	Execute Prog	Thread	2	VAR	9	C File C:\KMotionSrc\C Programs\ToolChange.c
M7	I/O Bit	Set bit	22	to	1	
M8	I/O Bit	Set bit	23	to	1	
M9	None					
S	DAC	Set DAC	0	scale	1	offset 0 min 0 max 1024

The M Codes 3 through 9 have functionality commonly used by most machine tools. These can also be used as custom or general purpose M codes in the same manner as the M100-M119 M codes with the exception of M5 and M9 which are automatically executed on Program Stop or Rewind.

M3 - Spindle On CW

M4 - Spindle On CCW

M5 - Spindle Off - Note Automatically called on Program Stop/Rewind

M6 - Tool Change (T variable is passed to the C program in the persist variable specified)

M3 - Spindle On CW

M4 - Spindle On CCW

M5 - Spindle Off - Note Automatically called on Program Stop/Rewind

The G Code Actions section of the Tool Setup Screen defines what action is to be performed when a particular G Code Command (Mostly M Codes) is encountered. The Action that can be performed can be one of several things:

- None
- Set or Reset one I/O Bit
- Set or Reset two I/O Bits
- Set a DAC to a variable's value (S parameter)
- Execute a C Program in the KMotion Control Board
- Execute a C Program in the KMotion Control Board + wait for it to terminate
- Execute a C Program in the KMotion Control Board + wait for it to terminate + resync Interpreter positions
- Execute a Windows Program

To specify a particular action first select the Action Type. Each Action Type requires a different number and type of parameters. Next fill in the appropriate parameters. The one and two bit I/O commands are inserted directly into the coordinated motion control buffer. In this way they are exactly synchronized with any motion before or after the I/O commands. This is useful in systems where a fast shutter or other operation is required at precise times relative to the motion.

The four Action Types are described below:

For **one I/O** bit specify the I/O bit number and the state 0 or 1 to set it to.

M7 Set bit to

For **two I/O** bits specify the I/O bit numbers and the state 0 or 1 to set each to. Often systems with two direction spindle control will require setting two outputs that control on/off and cw/ccw. This command is designed to handle those situations.

M3 Set bit to & bit to

For **DAC** specify the DAC (Digital to analog converter) channel number, how to scale and offset the associated variable, and the minimum and maximum allowed DAC settings. This command is primarily designed for use with the S (Spindle speed) G Code Command

S Set DAC scale offset min max

For **Execute Prog** specify the program Thread (1 through 7) where the program is to be downloaded and executed, a Persist Variable (1-99) that will be set before the program executes, and the name of the C Program to be Compiled, Downloaded, And Executed. If the Filename is left blank, then it will be assumed that a program has been previously downloaded and will just be re-executed. This method is very powerful in that anything that may be programmed in C may be invoked. See the KMotion documentation for information on writing C Programs for the KMotion Motion Control Board. There are a number of example C programs in the <Install Dir>\C Programs folder. The Example "Setup Gcode 4 axis.c" s an example which completely configures all necessary parameters on the KMotion Board to drive 4 stepping motors using KMotion's built in amplifiers.

Thread VAR C File >

Tool Setup Screen - M100 - M119

M Code	Action	Thread	VAR	C File
M100	Exec/wait/Sync	2	0	C:\KMotionSrc\C Programs\ProbeDirection.c
M101	I/O Bit	Set bit 44	to 1	
M102	I/O Bit	Set bit 45	to 0	
M103	None			
M104	None			
M105	None			
M106	None			
M107	None			
M108	None			
M109	None			
M110	None			
M111	None			
M112	None			
M113	None			
M114	None			
M115	None			
M116	None			
M117	None			
M118	Execute Prog	1	0	C:\KMotionSrc\C Programs\Setup Gcode 6 axis.c
M119	Execute PC			C:\WINDOWS\system32\notepad.exe

(Click on Image to Jump to related help)

The Tool Setup Screen allows *KMotionCNC* to be configured for a particular machine tool. Each machine tool is likely to have different motion resolution, speeds, and acceleration limits. It is also likely to have different I/O requirements with regard to Spindle control and such. Additionally a machine may have different initialization and homing requirements. *KMotionCNC* has a flexible mechanism for defining what type of action is to be performed for various M Codes and Custom Buttons.

G Code Actions M100-M119

This Tool Setup Screen Tab allows the user to define up to 20 custom general purpose M codes that can be used to change IO bits, or execute User C programs, or execute Windows Programs. The action to be performed is defined in the same manner as the [GCode Actions M3-M9](#).

Tool Setup Screen - User Buttons

The screenshot shows the 'Tool Setup Screen' window with the 'User Buttons' tab selected. The window contains the following fields and controls:

- Buttons:** INIT, HOME, bit0, bit1, and five empty slots.
- Fields for each button:**
 - Button Name:** Text input field.
 - Execute Prog:** Dropdown menu.
 - Thread:** Text input field.
 - VAR:** Text input field.
 - C File:** Text input field with a browse button (>).
 - I/O Bit:** Dropdown menu.
 - Set bit:** Text input field.
 - to:** Text input field.
- Buttons at the bottom:** Cancel, OK, Help.

(Click on Image to Jump to related help)

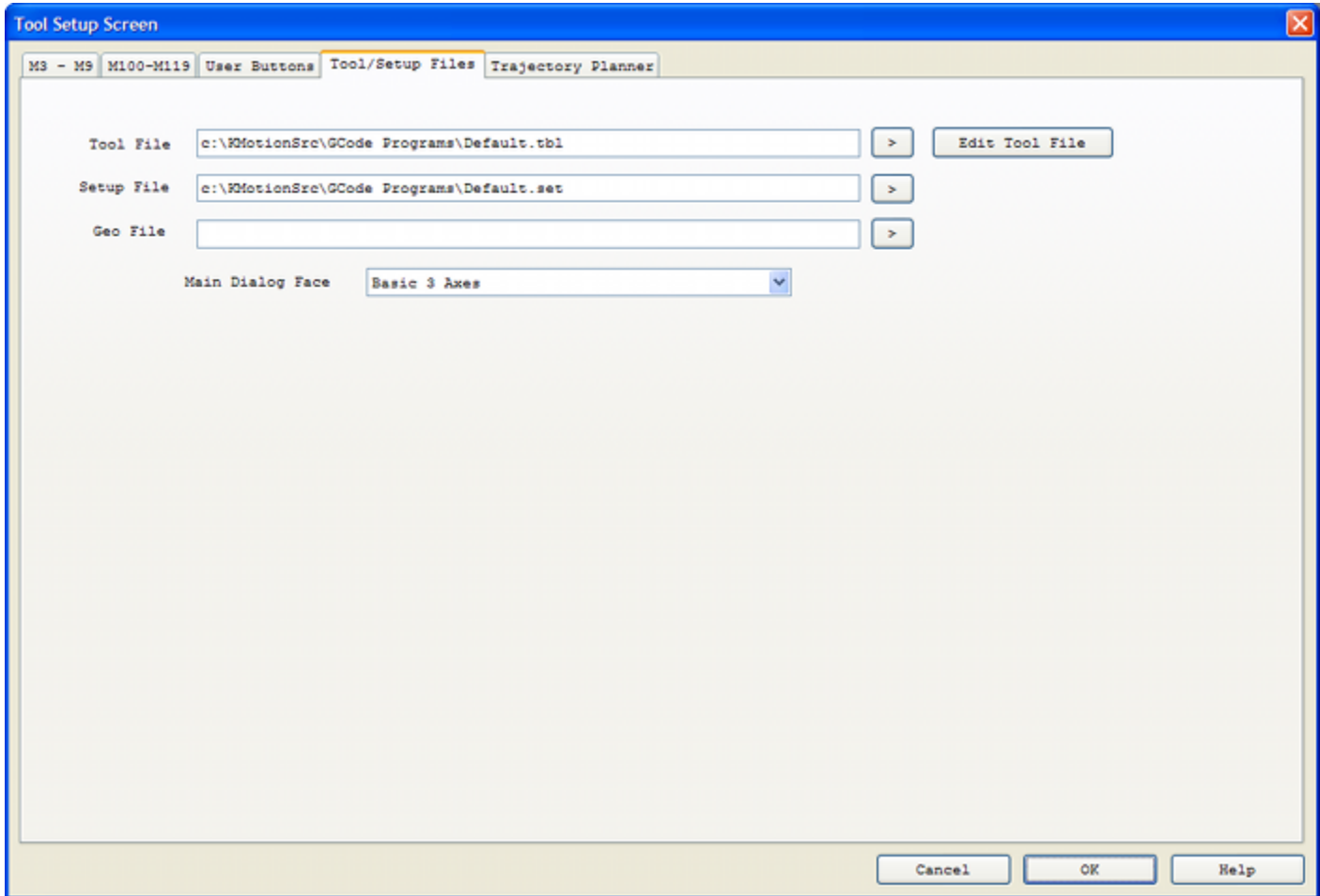
The Tool Setup Screen allows *KMotionCNC* to be configured for a particular machine tool. Each machine tool is likely to have different motion resolution, speeds, and acceleration limits. It is also likely to have different I/O requirements with regard to Spindle control and such. Additionally a machine may have different initialization and homing requirements. *KMotionCNC* has a flexible mechanism for defining what type of action is to be performed for various M Codes and Custom Buttons.

Custom Buttons

Button Name:	INIT			
Execute Prog	▼	Thread	1	VAR 0 C File C:\KMotionSrc\C Programs\Setup Gcode 6 axis.c >
Button Name:	HOME			
Execute Prog	▼	Thread	1	VAR 0 C File C:\KMotionSrc\C Programs\Home 6 axis.c >
Button Name:	bit0			
I/O Bit	▼	Set bit	144	to 0
Button Name:	bit1			
I/O Bit	▼	Set bit	144	to 1

Custom Button Actions function in exactly the same manner as the [G Code Actions](#) described on the M3-M9 Tab with the only difference being that they are invoked by the User pushing a button rather than by a command encountered within a G Code Program. There is an additional Parameter above the Action Type which is the Title to be placed on the Custom Button. The example above shows buttons defined with titles "INIT", "HOME", "bit0", and "bit1". Up to 10 buttons may be defined. Common uses for the Buttons are to invoke programs that initialize and/or home the machine. Any Button with an empty title field will cause the button to be hidden on the main *KMotionCNC* screen. See [here](#) for how these defined buttons will appear in the main *KMotionCNC* Screen.

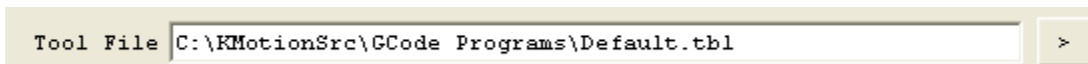
Tool Setup Screen - Files & Face



(Click on Image to Jump to related help)

The Tool Setup Screen allows *KMotionCNC* to be configured for a particular machine tool. Each machine tool is likely to have different motion resolution, speeds, and acceleration limits. It is also likely to have different I/O requirements with regard to Spindle control and such. Additionally a machine may have different initialization and homing requirements. *KMotionCNC* has a flexible mechanism for defining what type of action is to be performed for various M Codes and Custom Buttons.

Tool Table File

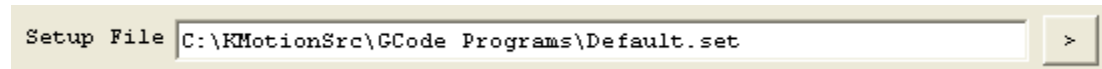


The Tool Table File specifies the disk text file which contains the table of tool definitions. In some cases the G Code Interpreter needs to know the length and diameter of the selected tool for tool path compensation. This file is used to define up to 99 tools. See also [Selecting Tools](#).

See below for an example Tool Table

POC FMS LEN DIAM COMMENT

1 1 0.0 0.0 first tool
 2 2 0.0 0.0
 3 3 1.0 0.5
 4 4 2.0 1.0
 32 32 0.0 0.0 last tool

Setup File

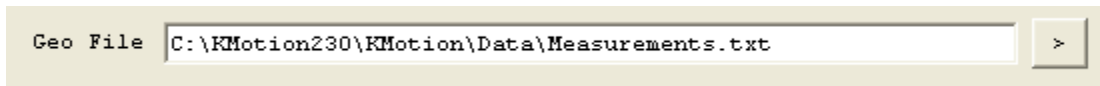
The Setup File specifies the disk

text file which contains the setup table for the G Code Interpreter. In some machine tools the Interpreter may require a special initialization state. Below is the default Setup file. Modifications to the setup file should not normally be required.

Attribute Value Other Possible Values

axis_offset_x 0.0 any real number
 axis_offset_y 0.0 any real number
 axis_offset_z 0.0 any real number
 block_delete ON OFF
 current_x 0.0 any real number
 current_y 0.0 any real number
 current_z 0.0 any real number
 cutter_radius_comp OFF LEFT, RIGHT
 cycle_r 0.0 any real number
 cycle_z 0.0 any real number not less than cycle_r
 distance_mode ABSOLUTE INCREMENTAL
 feed_mode PER_MINUTE INVERSE_TIME
 feed_rate 5.0 any positive real number
 flood OFF ON
 length_units MILLIMETERS INCHES
 mist OFF ON
 motion_mode 80 0,1,2,3,81,82,83,84,85,86,97,88,89
 plane XY YZ, ZX
 slot_for_length_offset 1 any unsigned integer less than 69
 slot_for_radius_comp 1 any unsigned integer less than 69
 slot_in_use 1 any unsigned integer less than 69
 slot_selected 1 any unsigned integer less than 69
 speed_feed_mode INDEPENDENT SYNCHED
 spindle_speed 1000.0 any non-negative real number
 spindle_turning STOPPED CLOCKWISE, COUNTERCLOCKWISE
 tool_length_offset 0.0 any non-negative real number
 traverse_rate 199.0 any positive real number

Geo File

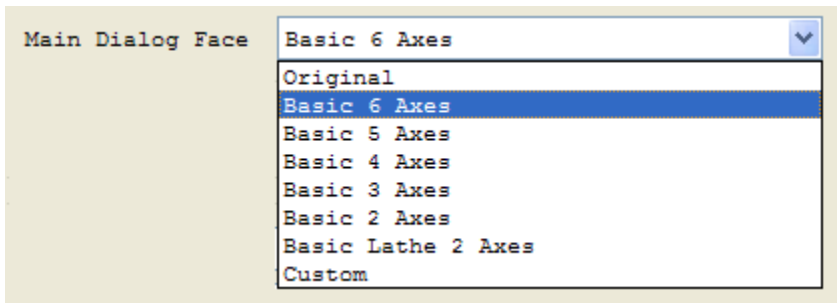


Geometric correction file. Allows correcting errors in the the x,y,z positions. The callibration procedure involves moving the tool tip to an xy array of positions. For example, a precision grid might be printed on a glass or Mylar plate. By Jogging the tool tip to each row and column grid point and recording the machine's x,y,z position, a table of machine coordinates that correspond to perfect coordinates may be obtained. The "measure" button on the main KMotionCNC Screen may be used to log these positions. If such a table is specified here, the system will correct machine coordinates by bilinear xy interpolation of this table. Z is corrected for flatness at the plane of z=0 only.

The table format is shown below. The first line specifies the number of rows and columns in the table. The second line specifies the delta x and delta y between gridpoints. An the remaing lines are row, column, x, y, z table entries.

```
5,5
1,1
0,0,-1.767822,-2.129132,-0.331770
0,1,-0.878572,-2.068192,-0.262691
0,2,0.036983,-1.979272,-0.234576
0,3,0.930314,-1.909252,-0.237177
0,4,1.836404,-1.805593,-0.236061
1,0,-1.774018,-1.155364,-0.219091
1,1,-0.882401,-1.073801,-0.179651
1,2,0.021794,-0.987806,-0.141523
1,3,0.918408,-0.885504,-0.130446
1,4,1.792961,-0.811495,-0.124016
2,0,-1.771149,-0.166872,-0.121132
2,1,-0.883159,-0.075746,-0.074570
2,2,0.007254,0.005231,-0.051105
2,3,0.889548,0.097292,-0.035076
2,4,1.762710,0.190093,-0.016807
3,0,-1.761705,0.784817,-0.086695
3,1,-0.885633,0.869580,-0.025417
3,2,0.001747,1.008045,0.017727
3,3,0.880585,1.108997,0.059038
3,4,1.742520,1.204385,0.058162
4,0,-1.729192,1.783193,0.021364
4,1,-0.871298,1.868678,0.054451
4,2,0.005017,1.979718,0.104414
4,3,0.878944,2.104289,0.141666
4,4,1.714383,2.182679,0.144980
```

Main Dialog Face



The Main Dialog Face selection selects the look of the main operation screen for KMotionCNC. Currently there are several available selections. The main difference is the number of axes, DROs and jog buttons. It is also possible to alter the dialogs by using a WindowsTM Resource Editor. Either Microsoft Visual Studio or a 3rd party freeware such as: <http://www.angusj.com/resourcehacker>.

Tool Setup Screen - Trajectory Planner

Tool Setup Screen

M3 - M9 | M100-M119 | User Buttons | Tool/Setup Files | **Trajectory Planner**

Trajectory Planner

Break Angle: 5 degrees Look ahead: 3 seconds Collinear Tolerance: 0.0001 in Corner Tolerance: 0.003 in Facet Angle: 0.5 degrees

Joystick/Jog

Jog Speeds: X 2 Y 1 Z 1 A 2 B 2 C 2 in/sec

Step Increments: 0.0001 0.001 0.01 0.1 1 10 ☐ Reverse R/Z

Axis Parameters

	Cnts/inch	Vel in/sec	Accel in/sec2
X	10000	16	30
Y	10000	16	30
Z	10000	16	30
A	100	10	10
B	100	10	10
C	100	10	10

☐ Lathe

Cancel OK Help

(Click on Image to Jump to related help)

The Tool Setup Screen allows *KMotionCNC* to be configured for a particular machine tool. Each machine tool is likely to have different motion resolution, speeds, and acceleration limits. It is also likely to have different I/O requirements with regard to Spindle control and such. Additionally a machine may have different initialization and homing requirements. *KMotionCNC* has a flexible mechanism for defining what type of action is to be performed for various M Codes and Custom Buttons.

Trajectory Planner

Trajectory Planner					
Break Angle	5	degrees	Look ahead	3	seconds
			Collinear Tolerance	0.0001	in
			Corner Tolerance	0.003	in
			Facet Angle	0.5	degrees

KMotionCNC contains a powerful Trajectory Planner. The Trajectory Planner utilizes a "break angle" concept to decide when a stop must be made. Vectors that are in the same direction within the "break angle" are traversed without stopping. When a "sharp" angle is detected a complete stop will be made before beginning on the next vector. The Break Angle Parameter allows the user to specify the angle in degrees that will be considered a "sharp" angle. *KMotionCNC* considers the change in direction in 3 dimensions (x,y,z ignoring a). The Trajectory Planner is capable of optimizing the acceleration and deceleration through many short (or long) vectors all of which may have different acceleration and velocity limitations.

The Trajectory Planner also has a "lookahead" parameter. With *KMotionCNC* the G Code Program itself, the G Code Interpreter, and the Trajectory Planner all reside within the PC running Microsoft Windows™. Since the Microsoft Windows™ is *not* a real-time OS, a certain amount of motion must be buffered in the motion controller to handle the cases where the PC program doesn't have a chance to execute for a while. These time periods are typically very short (a few milliseconds), but in some extreme cases may occasionally be as long as one or several seconds. The worst case is often a factor of the hardware (disk network adapters, etc) and associated drivers that are loaded into the system. The lookahead parameter is used to determine how much motion, in terms of time, should be downloaded to the motion controller *before* actual motion is initiated. Furthermore, after motion has begun, the lookahead parameter is used to pause the trajectory planner to avoid having the Trajectory Planner get too far ahead of the actual motion. The disadvantage of having the Trajectory Planner get too far ahead is that if the User decides to override the feed rate, since the motion has already been planned and downloaded, the rate will not be changed quickly. A value of 3 seconds is very conservative on most systems. If more responsive feed rate override is desirable, an experiment with a smaller value might be made.

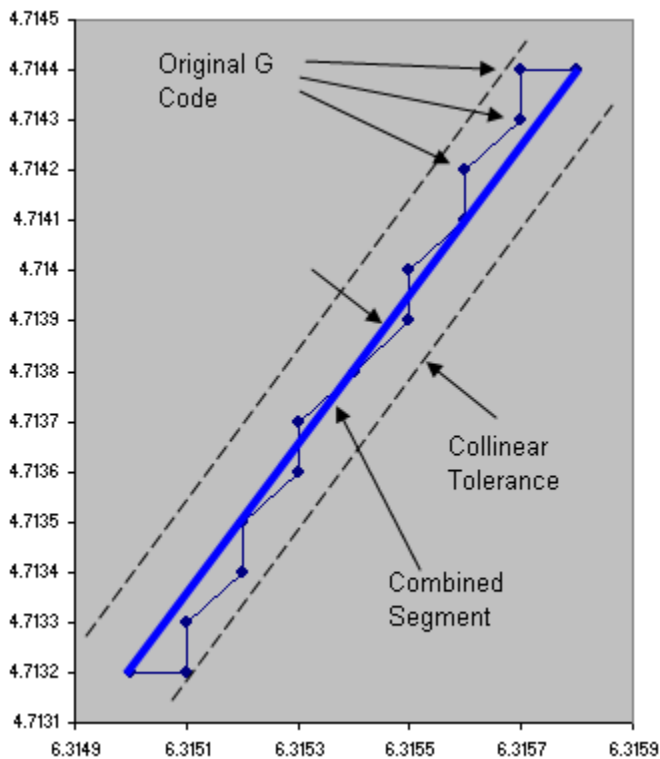
"Collinear Tolerance" allows GCode blocks that are nearly linear, to be combined into a single linear segment to allow faster and smoother motion. Often GCode programs generated by 3D CAD systems contain very small vectors that appear jagged or noisy due to numerical round off errors. See below for an example of a small GCode fragment from an intended circle. The 0.3 inch diameter circle was divided into 10,000 sides each only 0.0001 inches in length. Intuitively one would think this would result in an extremely smooth contour. Ironically, rounding off to 4 decimal digits introduces "noise" that results in sharp angles and each axis being required to repeatedly stop and start. Even with quite high acceleration, stopping and starting over short distances results in extremely low average speed and most likely a rough surface. A combined segment shown below in blue will result in a faster and smoother motion.

Segments are combined as long as all intermediate waypoints do not deviate from the combined segment by more than the allowed collinear tolerance. Setting the collinear tolerance to zero will disallow any segments from being combined.

```

N70 G90 G54 G0 X6.315 Y4.7132 Z1.
N100 G1 X6.3151 F60.
N110 Y4.7133
N120 X6.3152 Y4.7134
N130 Y4.7135
N140 X6.3153 Y4.7136
N150 Y4.7137
N160 X6.3154 Y4.7138
N170 X6.3155 Y4.7139
N180 Y4.714
N190 X6.3156 Y4.7141
N200 Y4.7142
N210 X6.3157 Y4.7143
N220 Y4.7144
N230 X6.3158

```



For a description of the *Corner Rounding* feature and Corner Tolerance and Facet Angle see [here](#).

Jog Speeds

Jog Speeds X Y Z A B C in/sec

Defines the Jog Speeds for both the Jog Buttons and any attached Gamepad controller. These speed are the maximum Jog speed which is the double arrow jog button or the GamePad joystick

fully depressed. See Also [Jog Buttons](#). The Reverse R/Z may be selected if the GamePad Z motion is reversed on a particular GamePad device.

Step Increments

Step Increments ☐ Reverse R/Z

Defines the step size distances for the Step Buttons that are displayed on the main *KMotionCNC* Screen. Setting a step size to zero will hide the size selection. See Also [Jog Buttons](#).

Axis Motion Parameters

The Axis Motion Parameters define the scale, maximum feed velocities, and maximum feed accelerations for each of the six axis.

Axis Parameters			
	Cnts/inch	Vel in/sec	Accel in/sec ²
X	<input type="text" value="1000"/>	<input type="text" value="16"/>	<input type="text" value="10"/>
Y	<input type="text" value="500"/>	<input type="text" value="16"/>	<input type="text" value="10"/>
Z	<input type="text" value="200"/>	<input type="text" value="10"/>	<input type="text" value="10"/>
A	<input type="text" value="100"/>	<input type="text" value="10"/>	<input type="text" value="10"/>
B	<input type="text" value="100"/>	<input type="text" value="10"/>	<input type="text" value="10"/>
C	<input type="text" value="100"/>	<input type="text" value="10"/>	<input type="text" value="10"/>

The first parameter is the axis's scale in counts/inch. For the example value of 100 shown, *KMotionCNC* will command the motion controller to move 100 counts for each inch of motion in the G Code Program. This value is always in counts/inch regardless of the units used in the interpreter. *KMotionCNC* will automatically perform any conversions.

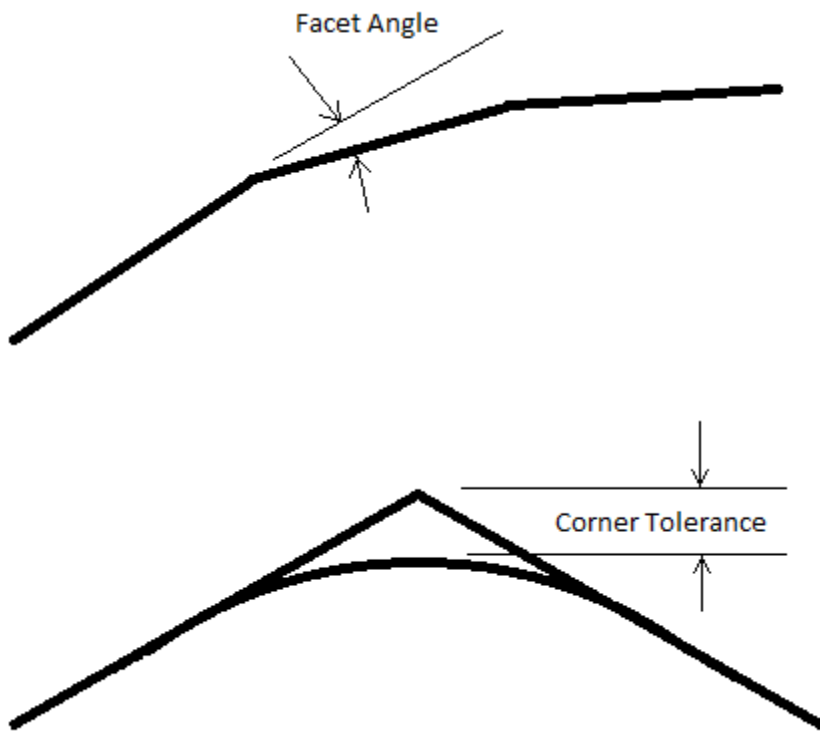
The second parameter is the maximum allowed feed rate for the axis in inches/sec. Note that the G Code Interpreter Feed Rate is defined in inches per minute or (mm per minute) so be aware of the different time units. These are *maximum* feed rates for each axis. If a vector tool motion at a particular feed rate has any component velocity that exceeds the corresponding axis's maximum velocity, the feed rate for the vector will be reduced so that all axes remain at or below their maximum allowed velocity.

The third parameter is the maximum allowed acceleration for the axis in inches/sec². The G Code Language has no provisions for specifying acceleration rates. Therefore the acceleration (and deceleration) along a vector used will always be the largest acceleration such that each axis's acceleration is at or below the specified limit.

The velocity and acceleration limits apply only to linear and circular feed motions (G1, G2, G3). Rapid motions (G0) use the settings in the motion controller (velocity, acceleration, and Jerk) to move each axis independently from one point to another (which is likely not to be a straight line). To change the speed of Rapid motions change the configuration settings in the motion controller.

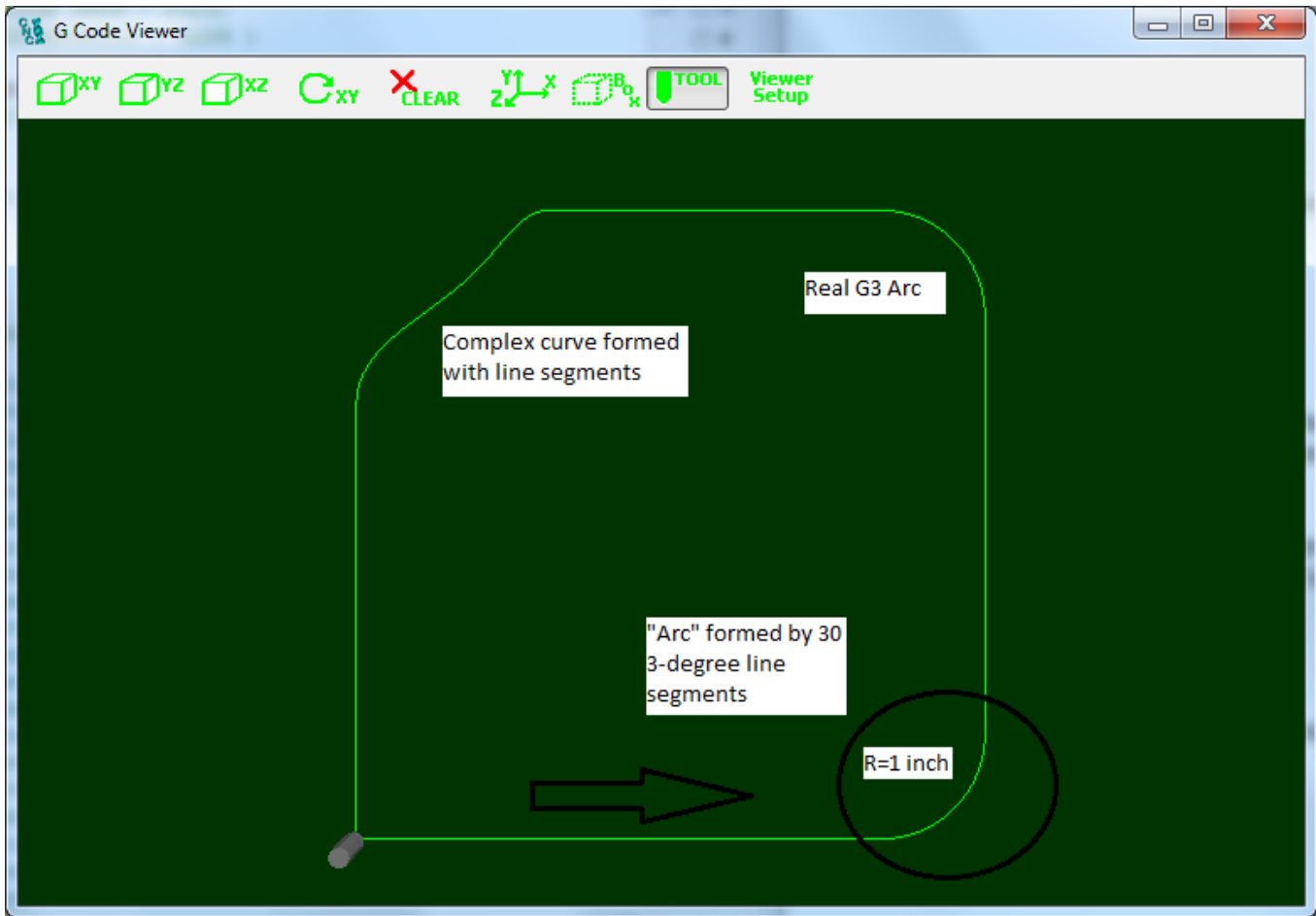
Trajectory Planner Corner Rounding Settings

The KMotionCNC Trajectory planner contains two parameters (Facet Angle and Corner Tolerance) that can be used to smooth paths generated by line segments in the GCode. Although shown below as 2D paths this smoothing applies for 3D paths as well. Standard GCode and many CAD systems do not support arcs in arbitrary 3D space so small line segments are typically used to define the 3D path. Facet Angles in the original GCode data that are less than the specified Break Angle (which are considered "small" and will not cause a motion stop) will be re-faceted with smaller line segments along a curve to have Facet Angles smaller than the specified amount. The new line segments will be placed along the largest arc that will not deviate more than the specified Corner Tolerance or consume more than 1/2 of the segment length. If no Corner Rounding is desired the Corner Tolerance can be set to zero.

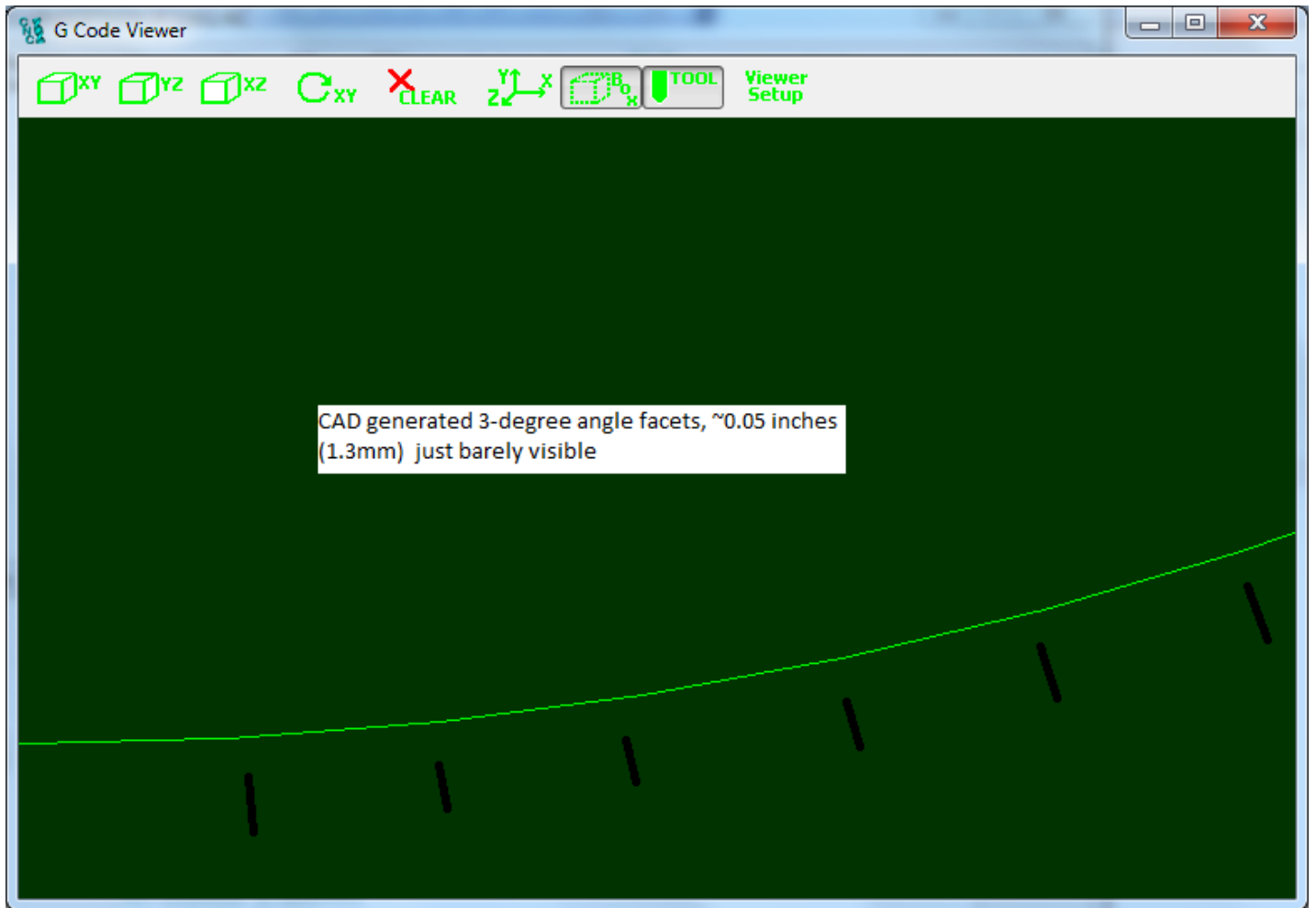


Shown below is an example motion path where the bottom right arc is formed by 30 line segments. Normally a simple arc could be specified but for test purposes it was created with 30 3 degree angle facets forming the 90 degree turn. The tests below were performed simulating a fairly high performance system with 30 in/sec² acceleration in both X and Y. The Gcode used can be downloaded [here](#).

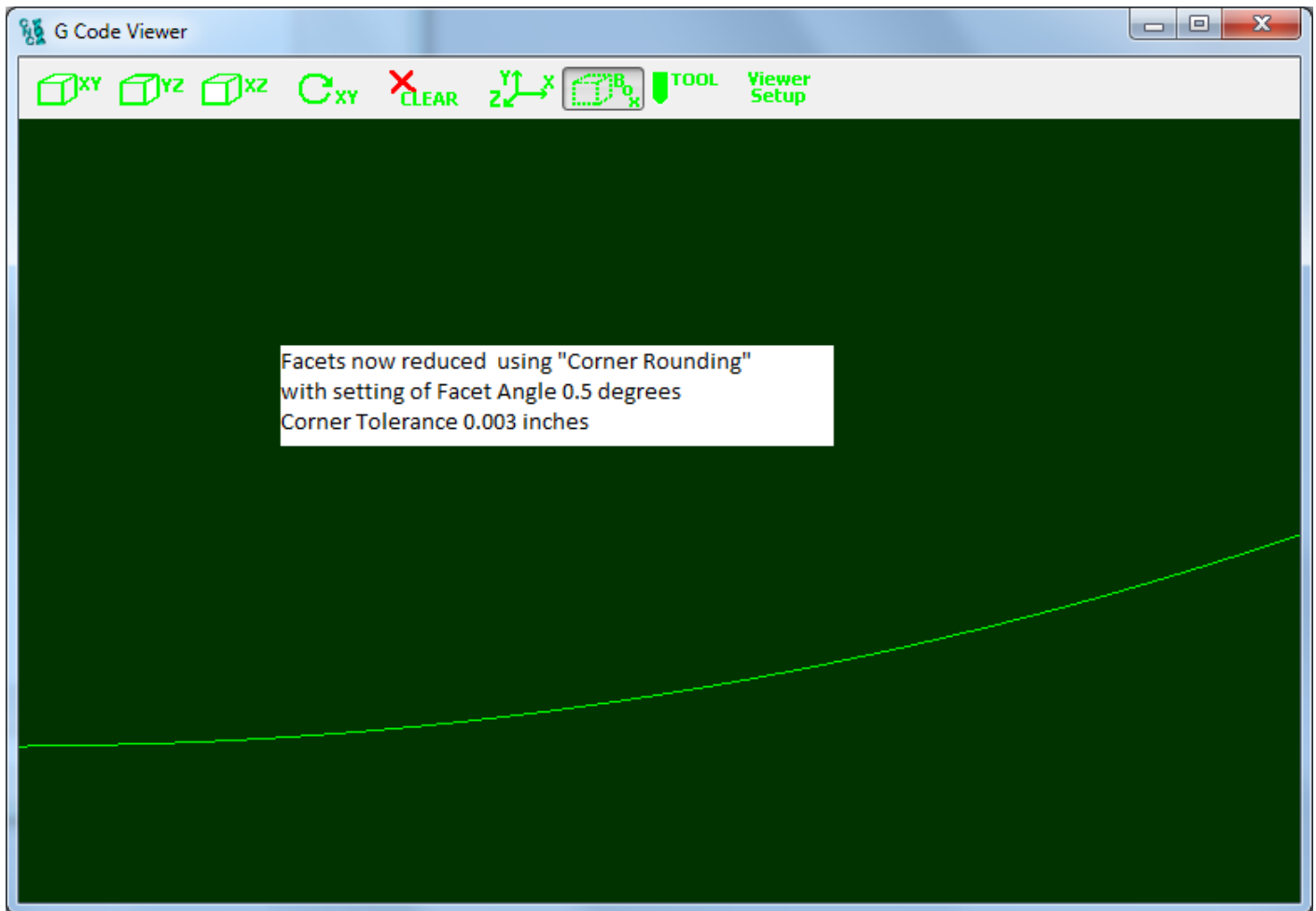
Here is the path Overview.



Notice the circled region of the path looks smooth, but when zooming in as shown below the 3-degree angle facets are just barely visible. This resolution is typically used to provide a reasonably smooth path without making the file size abnormally large.



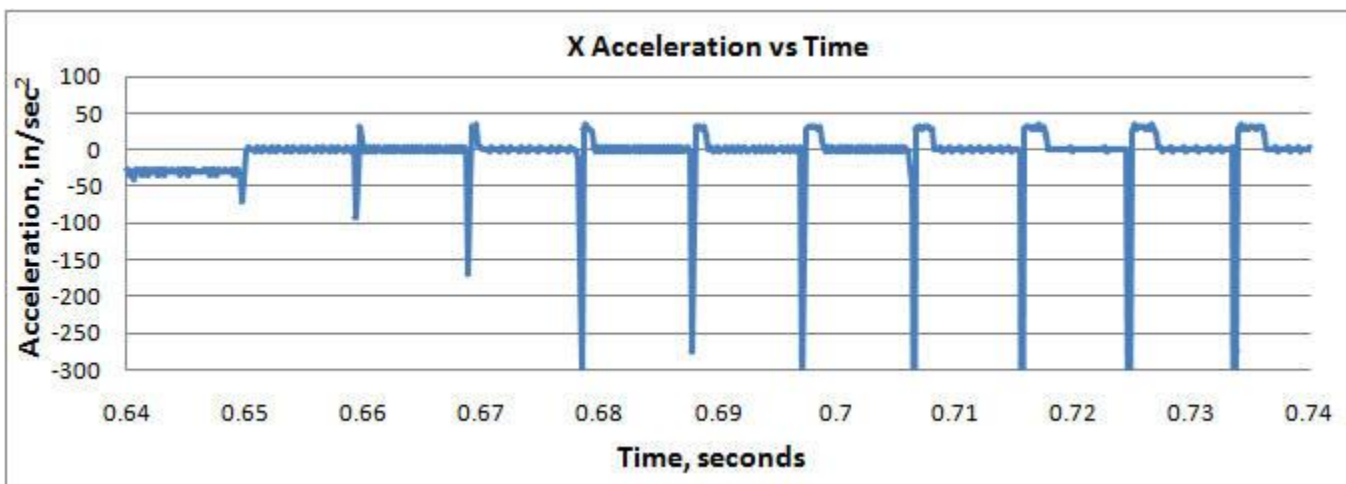
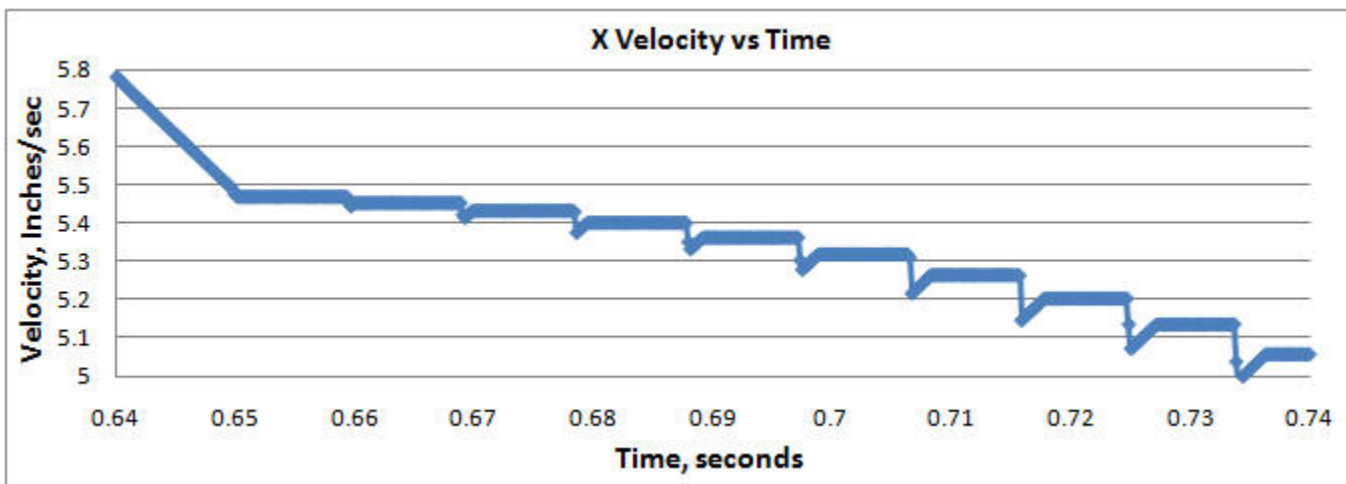
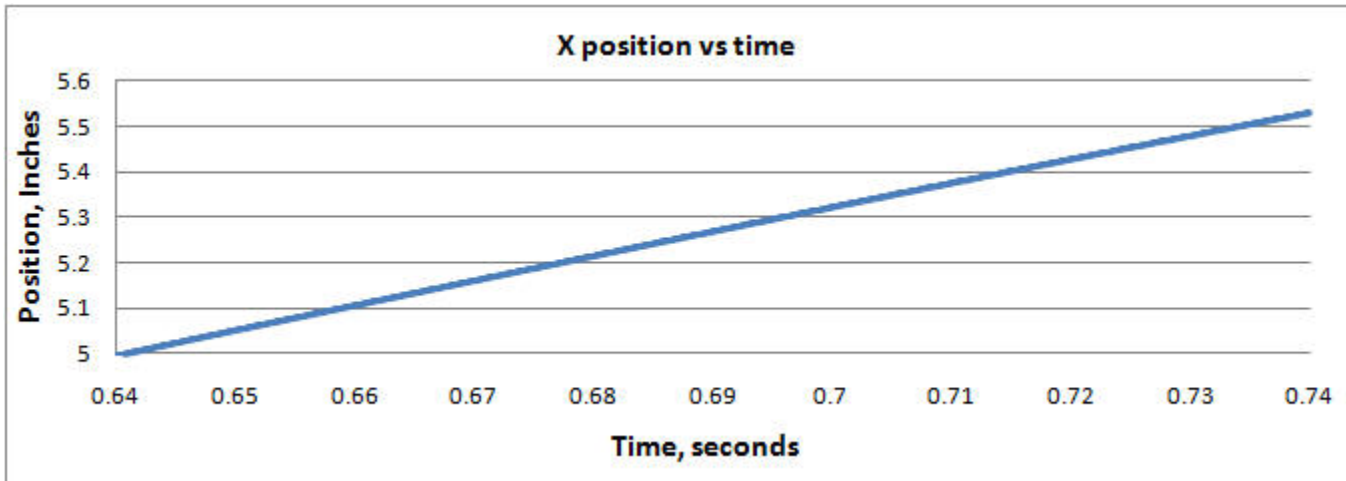
In the plot below the Corner Rounding is enabled and the facets are now reduced in size and form 0.5 degree angles that can now no longer be seen.



To see more clearly the improvement the X Axis motion was captured in real-time (180us sample rate) and plotted. From the position the Velocity and Acceleration were also computed and plotted.

First the original captured motion X axis position, velocity, and acceleration vs. time through the original 3-degree facets with no corner rounding.

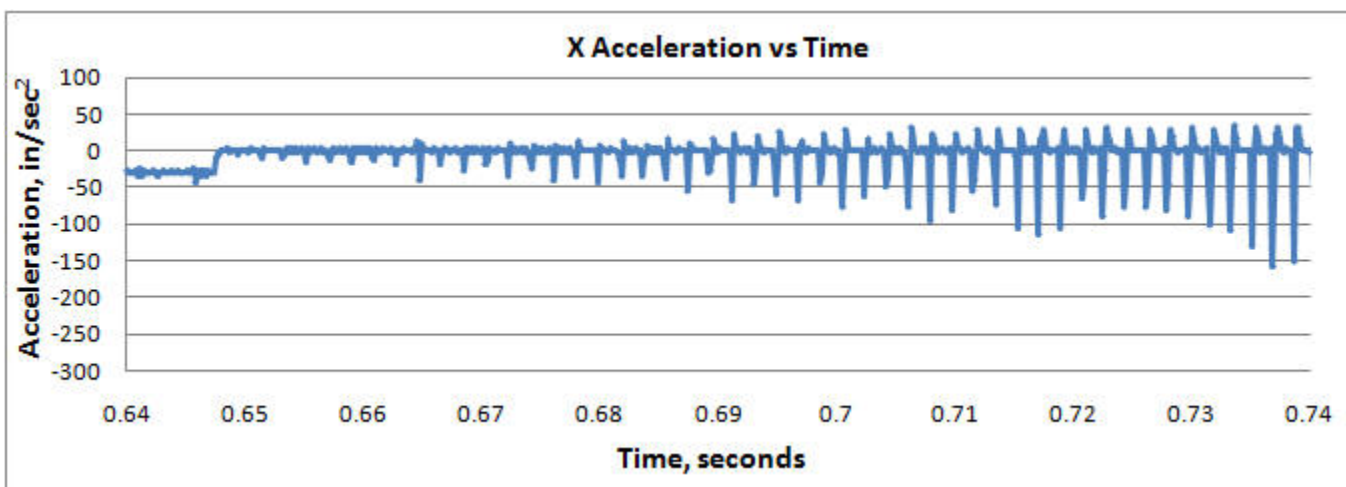
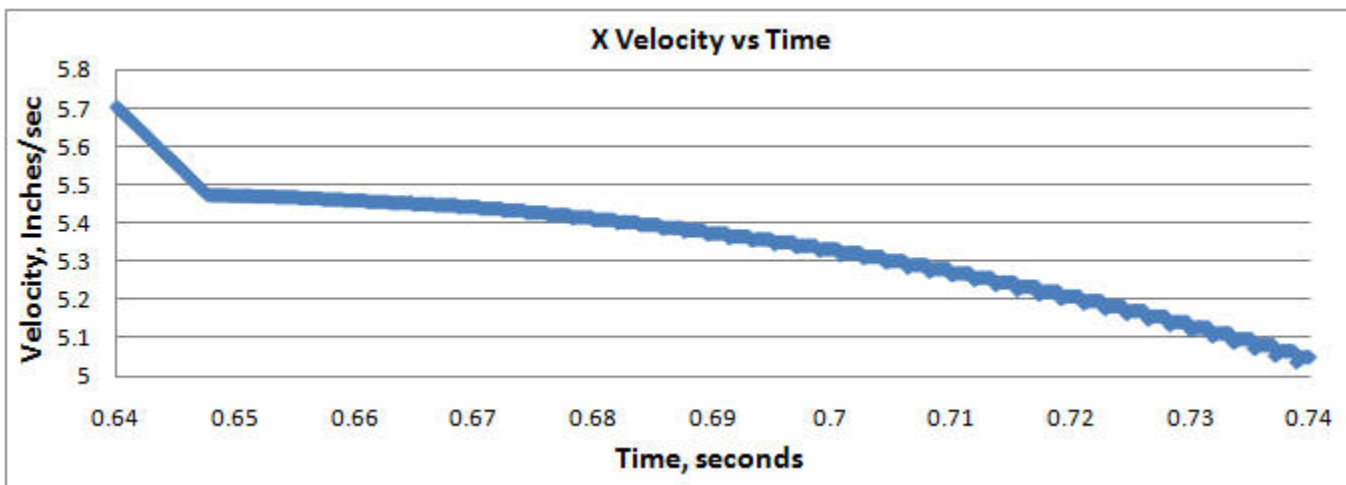
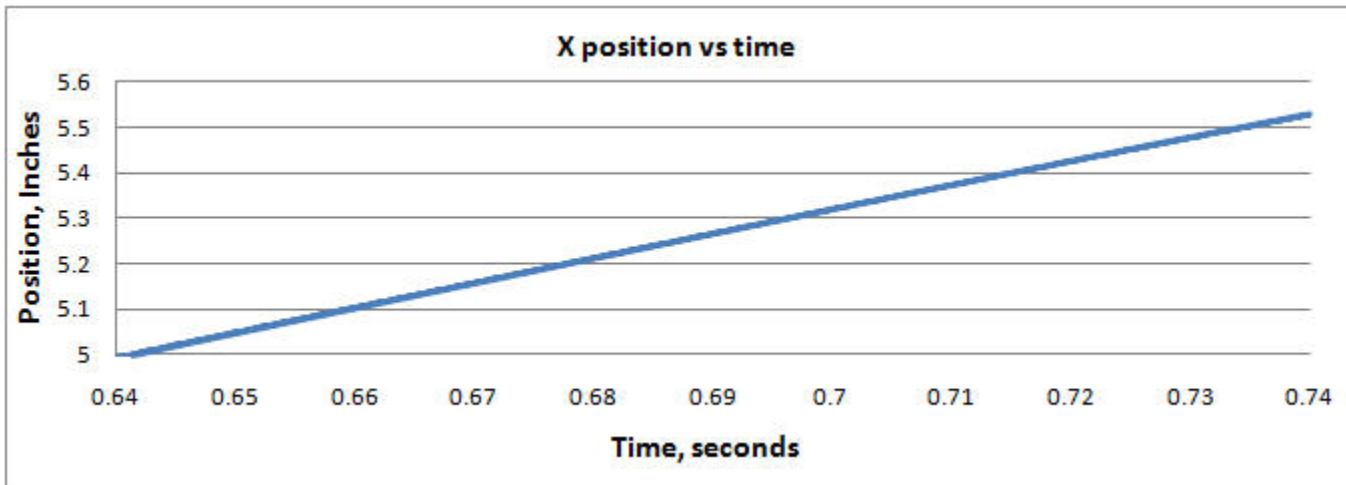
Note that the velocity plot has an odd shape for several reasons. At constant velocity the facet angle change causes the x velocity to drop while the y velocity increases, but since we are accelerating through the curve (because a more diagonal direction can make use of the increased (combined) acceleration of both axes) the velocity ramps up at the max acceleration toward the beginning of each facet. The programmed feed rate is 400ipm (6.67ips) but the speed is acceleration limited by the curvature of the path, hence the deceleration to ~ 5.5ips on the left as we enter the curve.



Now using the Trajectory Planner settings shown below. Notice the velocity is much smoother and the Acceleration is less Jerky.

Trajectory Planner

Break Angle degrees Look ahead seconds Collinear Tolerance in Corner Tolerance in Facet Angle degrees



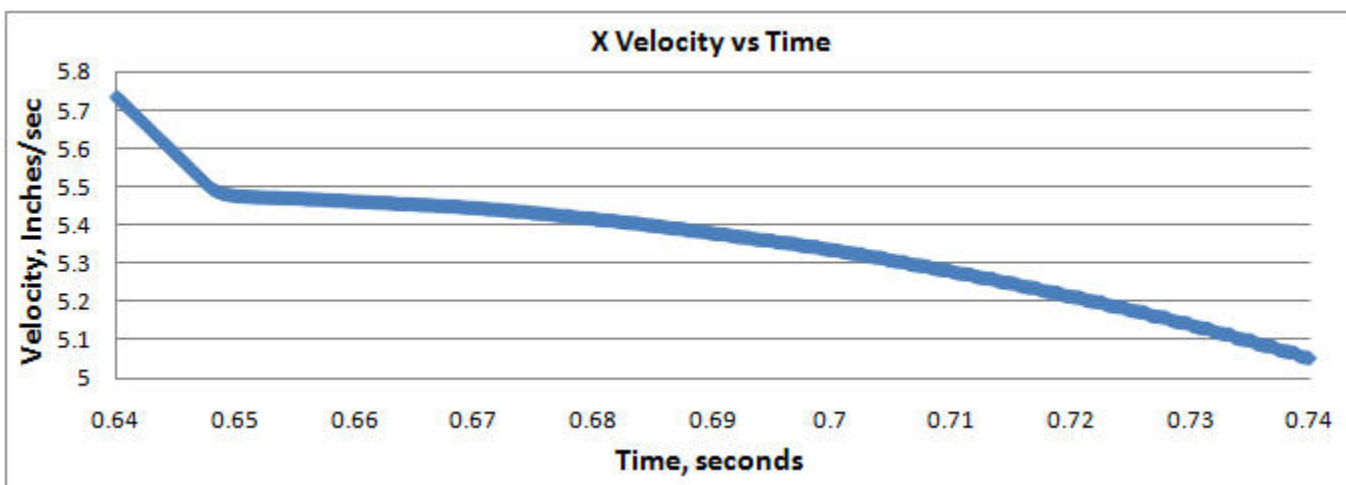
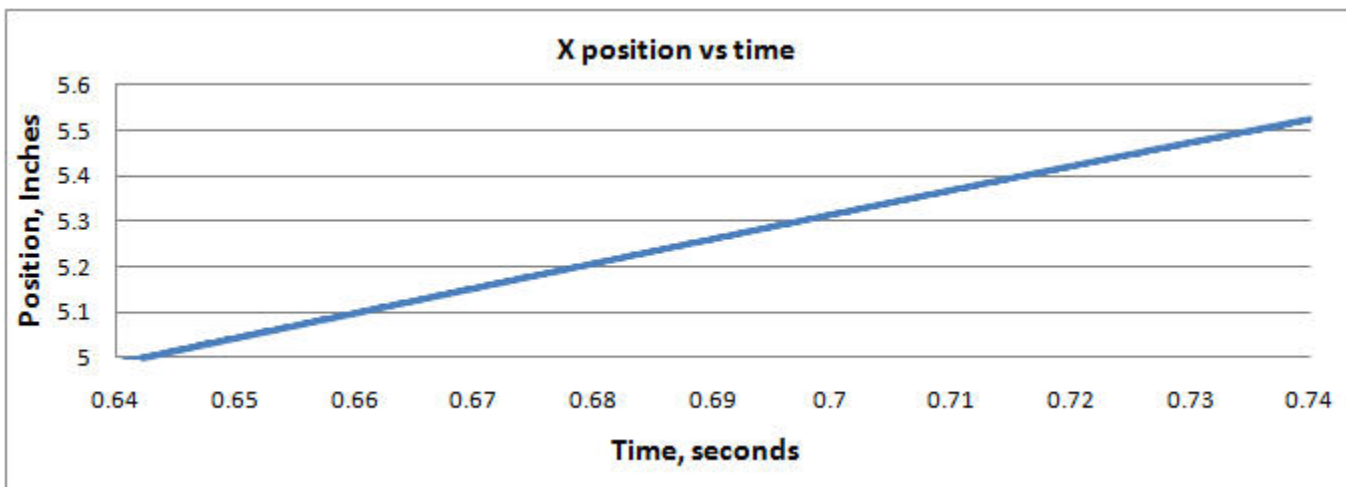
For still further motion smoothing an additional KFLOP feature can be used. A low pass filter can be applied to the output of the coordinated motion path.

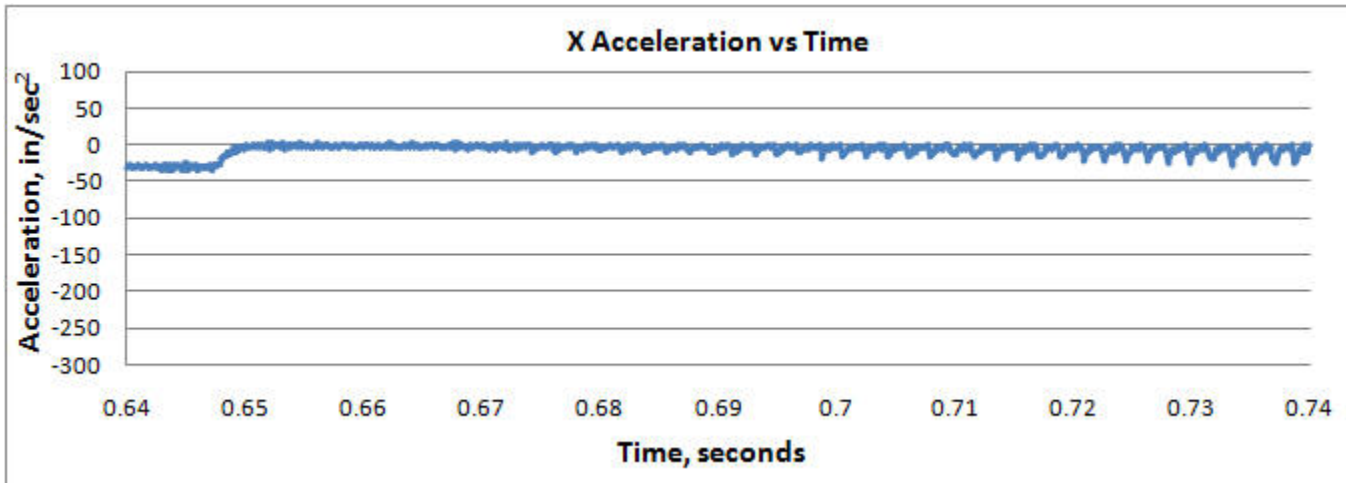
The Low Pass Filter will be applied to all axes of coordinated motion (up to 8) by setting the KLP coefficient within KFLOP. Currently a C Program must be used to set this global parameter.

To compute an appropriate coefficient from a time constant Tau in seconds use the formula $KLP = \exp(-TIMEBASE/Tau)$ as shown below

```
#include "KMotionDef.h"
main()
{
double Tau = 0.001; // seconds for Low Pass Filter Time Constant
KLP = exp(-TIMEBASE/Tau);
printf("Tau=%f KLP=%f\n", Tau, KLP);
}
```

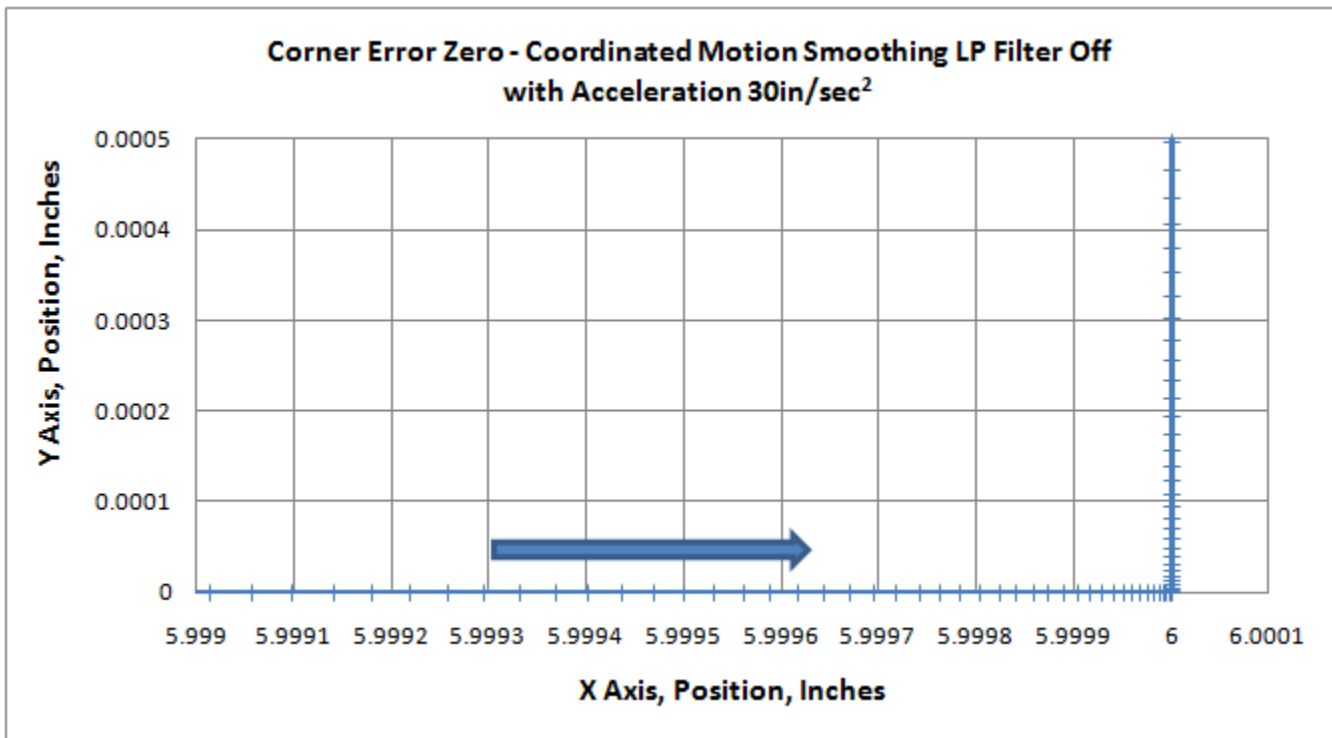
Note the Velocity and Acceleration plots are even smoother. A low pass time constant Tau = 1 millisecond was used.



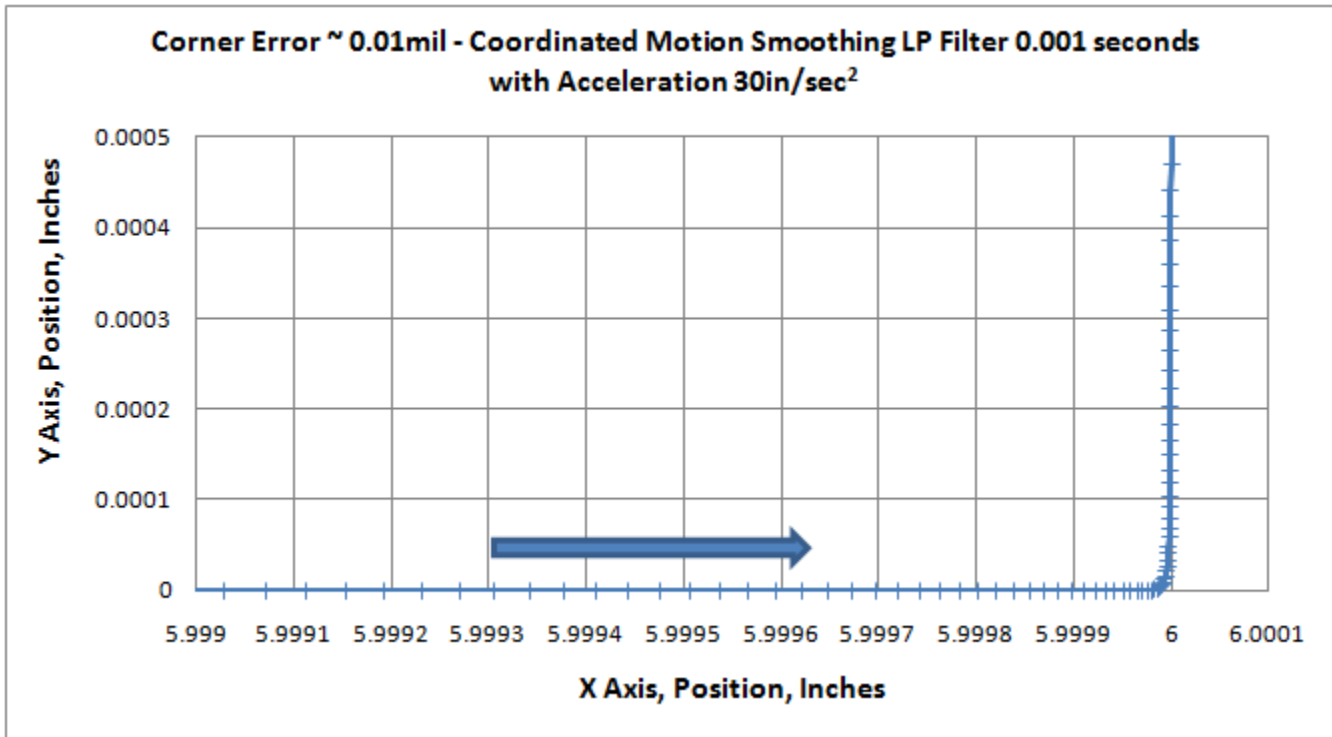


The disadvantage associated with low pass filtering of the trajectory is a potential lag in the commanded positions which may cause small path errors. The plots shown below show that a Low Pass Filter setting of 1 millisecond will be insignificant for most systems. A nearly worst case 90-degree angle with max deceleration on the X axis, followed by max acceleration on the Y axis are shown below.

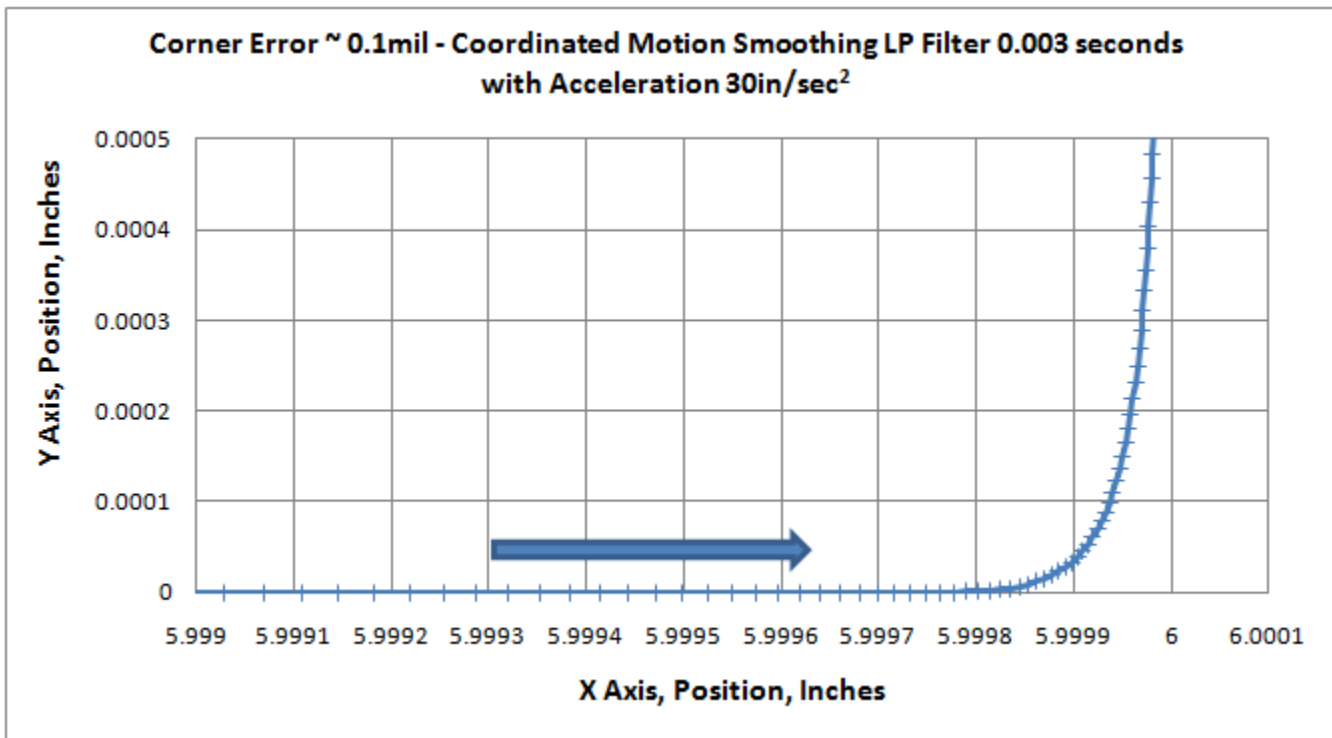
The first case is captured with no Low Pass Filtering. Note a perfectly square corner. Each tick mark shows a captured 180us sample point.



The case below is with 1 millisecond of filtering with a path error of ~ 0.01 mil (0.25um)



The case below is with 3 milliseconds of filtering with a path error of ~ 0.1 mil (2.5 μ m)



Mach3 Plugin



+



KMotion/KFlop Mach3

Mach3 is a popular CNC program available for purchase through [ArtSoftControls](http://www.artsoftcontrols.com).

Traditionally Mach3 has relied on little hardware support and performed low level motion and even stepper motor step pulses directly by the PC. The pulses were output using the PC's parallel port. This required a special Microsoft Windows Kernel driver with a high interrupt rate and was limited to 50~100K steps per second.

Mach3 has the capability for adding "plugins" that allow additional functionality and hardware support. **DynoMotion** has developed a plugin that allows Mach3 to work with a **KMotion** or **KFLOP** Motion Controller. The offloads most of the real-time requirements from the PC, allows USB connectivity, much higher step rates, and allows easily adding other motor types including brushless servos.

This is an overview of the overall setup process of getting Mach3 and KMotion to play together.

1. Install Mach3
2. Install KMotion
3. Within KMotion configure/tune your motors
4. Within KMotion create a configuration and initialization program
5. Within Mach3 - configure plugin (enter name of KMotion Init program)
6. Within Mach3 - configure motor tuning (set resolution, speeds, and acceleration)
7. Within Mach3 - configure IO Bits.

Note: when using encoder feedback see also: [Mach3 Encoder Setup](#)

Note: for configuring probing see also: [Mach3 G31 Probe Setup](#)

Note: for information regarding operating steppers in closed loop see also: [Closed Loop Steppers](#)

Note: For passing Parameters see also: [Passing DROs](#)

Note: For Rigid Tapping see also: [Mach3 Rigid Tapping](#)

1. Install Mach3

Mach3 should be installed *before* KMotion so that the KMotion installation program can copy the necessary files (Dynomotion.dll) into the Mach3 plugin directory. The KMotion installation will also add a registry entry under "App Paths" so that Mach3.exe will have access to the necessary KMotion DLL libraries, programs, and data files

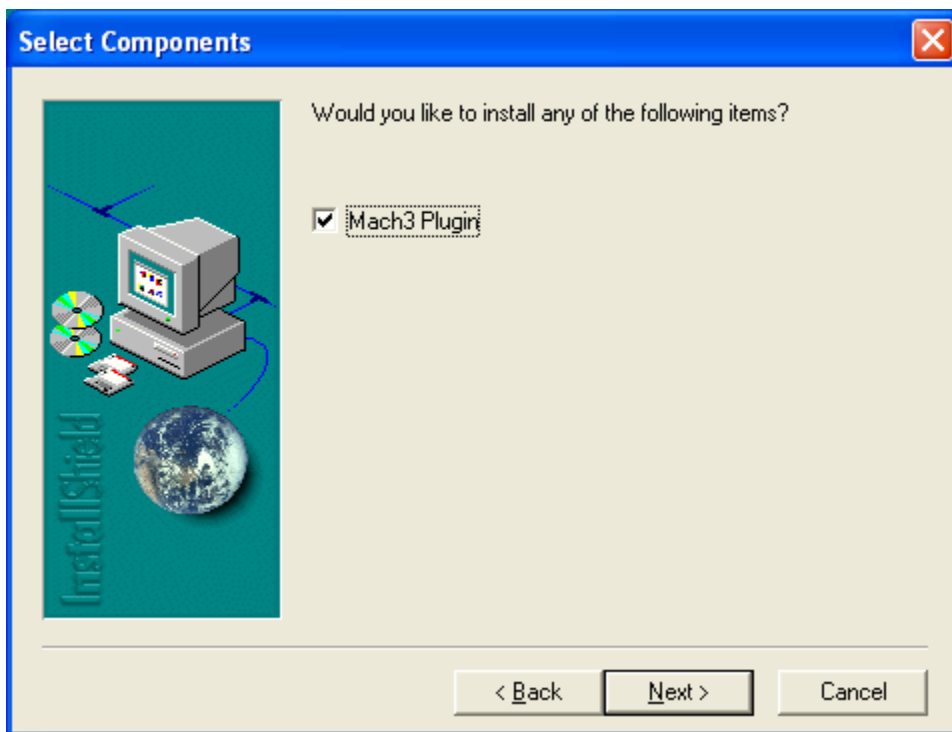
It is not necessary to load the Mach3 Kernel driver.

If KMotion has already been installed, it should be re-installed to make the necessary links between the programs.

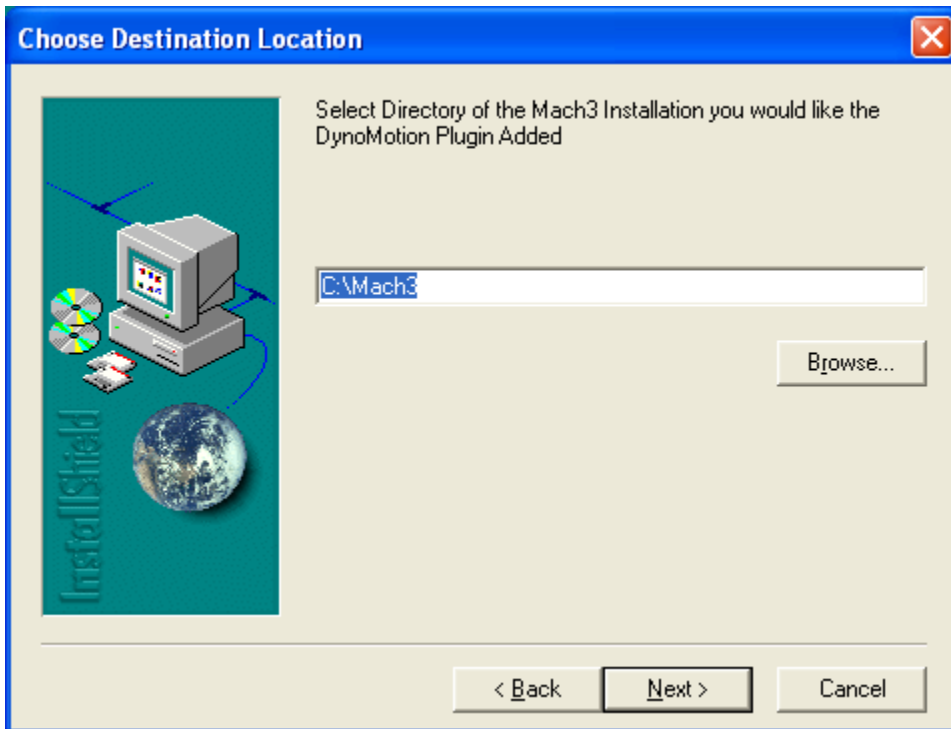
2. Install KMotion

At the end of the KMotion installation an option to install the Mach3 Plugin will be displayed.

Check the Mach3 Plugin option and select Next.



You will be prompted to enter the directory where Mach3 was installed.



A check will be made that a valid Mach3.exe exists in that directory, and the Dynomotion.dll will be copied to the plugins subdirectory.

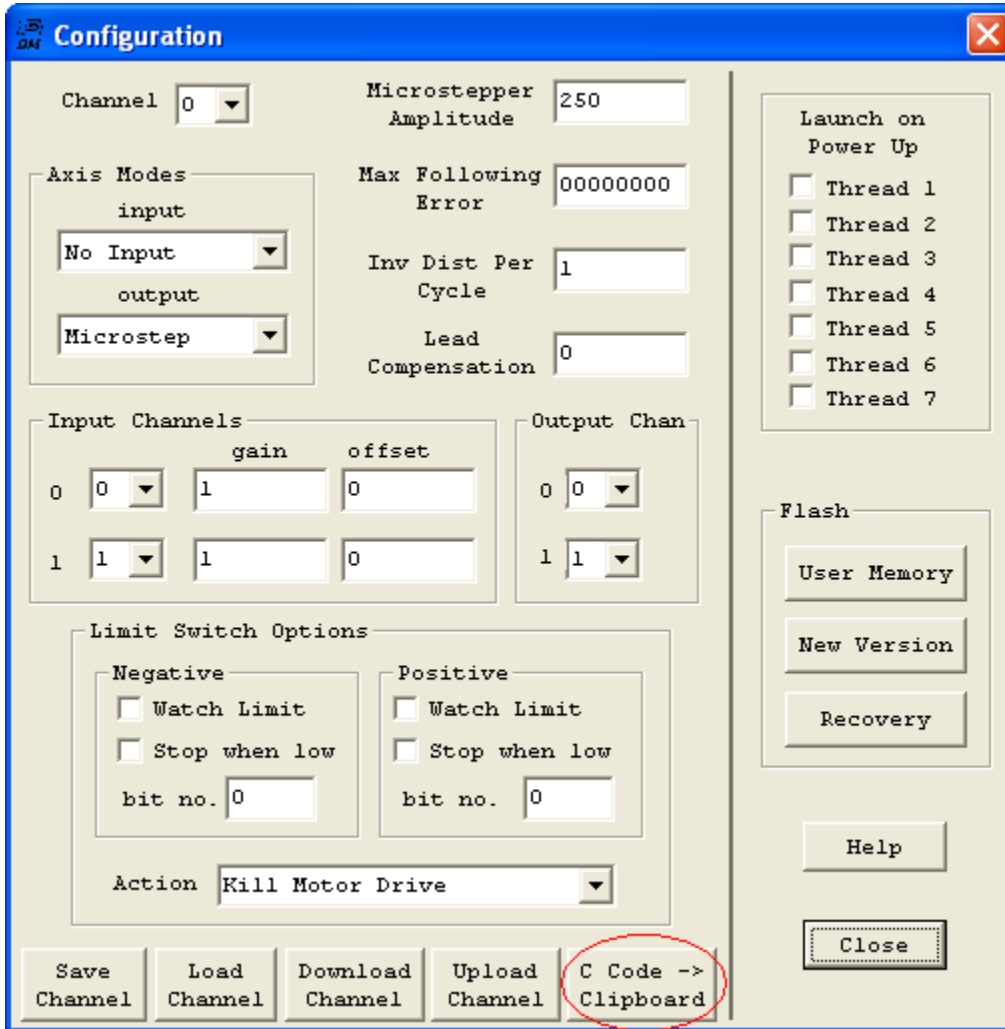
3. Within KMotion configure/tune your motors

Follow the normal procedures to use the KMotion Executive program to configure and tune all of your motor axes. The simplest method to configure the system is to enter and change values in the various KMotion Executive Screens, download them to the KMotion Board, and test and tune for correct operation using the Step Response Screen as well as Console commands.

4. Within KMotion create a configuration and initialization program

After each axis is functioning properly, the configuration for that axis can be copied to the clipboard as a valid C program code.

See the circled button below.



C Code to configure axis 0 might look like the code shown below. Detailed knowledge of C programming is not required to paste these into a User Program. The configuration for each axis should be pasted into a single program. Later this program may be executed at any time to completely configure the KMotion Board. The KMotion Executive program would not be needed to do this. The RESET button from within Mach3 will be configured to execute this program.

```
ch0->InputMode=ENCODER_MODE;
ch0->OutputMode=MICROSTEP_MODE;
ch0->Vel=100.000000;
ch0->Accel=1000.000000;
ch0->Jerk=10000.000000;
ch0->P=1.000000;
ch0->I=0.000000;
ch0->D=0.000000;
ch0->FFAccel=0.000000;
ch0->FFVel=0.000000;
ch0->MaxI=200.000000;
ch0->MaxErr=200.000000;
ch0->MaxOutput=200.000000;
```



```

ch0->DeadBandGain=1.000000;
ch0->DeadBandRange=0.000000;
ch0->InputChan0=0;
ch0->InputChan1=1;
ch0->OutputChan0=0;
ch0->OutputChan1=1;
ch0->LimitSwitchOptions=0x0;
ch0->InputGain0=1.000000;
ch0->InputGain1=1.000000;
ch0->InputOffset0=0.000000;
ch0->InputOffset1=0.000000;
ch0->invDistPerCycle=1.000000;
ch0->Lead=0.000000;
ch0->MaxFollowingError=1000000000.000000;
ch0->StepperAmplitude=250.000000;

ch0->iir[0].B0=1.000000;
ch0->iir[0].B1=0.000000;
ch0->iir[0].B2=0.000000;
ch0->iir[0].A1=0.000000;
ch0->iir[0].A2=0.000000;

ch0->iir[1].B0=1.000000;
ch0->iir[1].B1=0.000000;
ch0->iir[1].B2=0.000000;
ch0->iir[1].A1=0.000000;
ch0->iir[1].A2=0.000000;

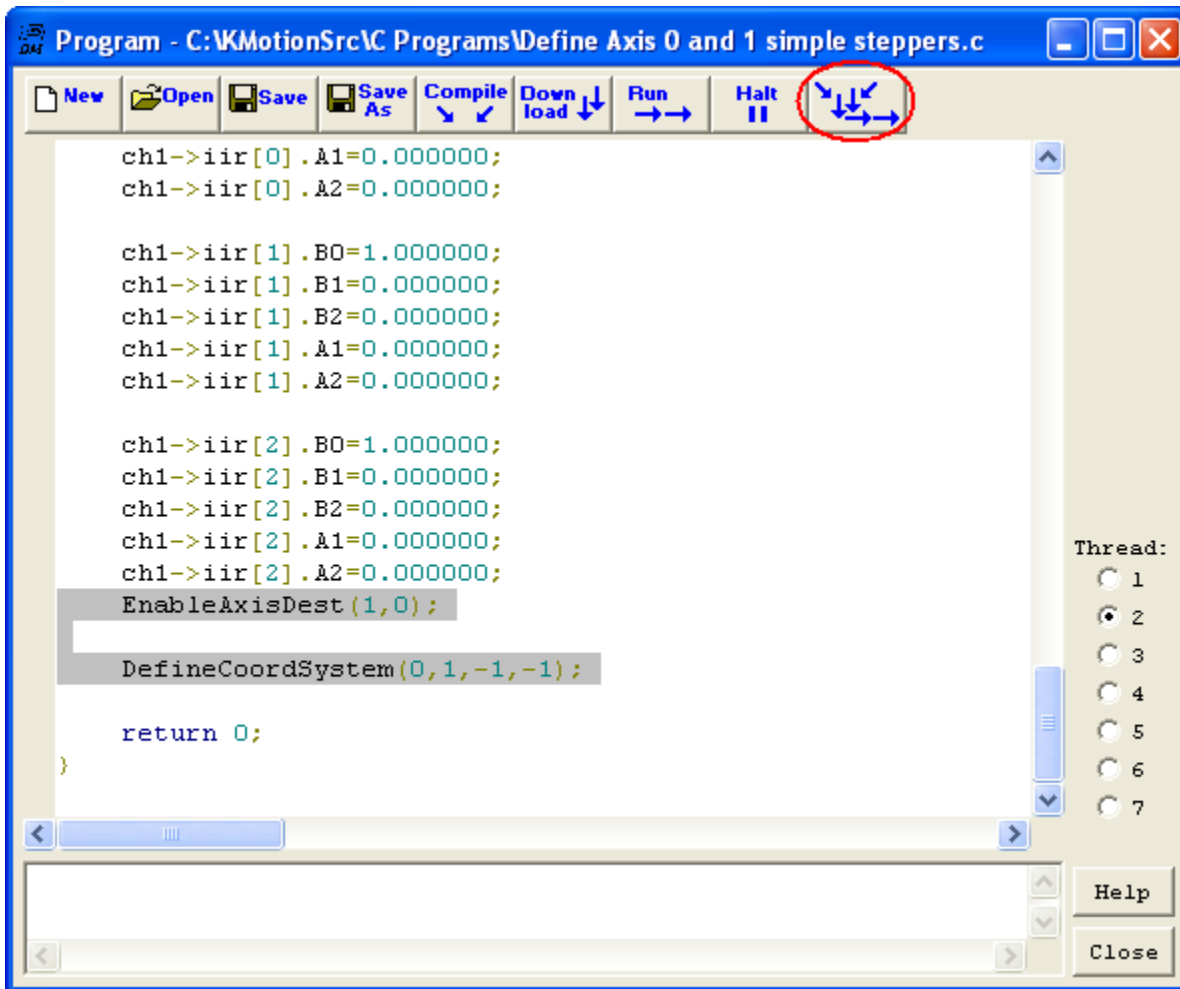
ch0->iir[2].B0=1.000000;
ch0->iir[2].B1=0.000000;
ch0->iir[2].B2=0.000000;
ch0->iir[2].A1=0.000000;
ch0->iir[2].A2=0.000000;

```

Besides C code to configure each axis, other commands such as those shown highlighted below may be used to enable each axis (and set the destination) and to define which axis channels are in the coordinated motion system. The `DefineCoordSystem(0,1,-1,-1);` statement defines a two axis coordinate system where X is axis channel 0, Y is axis channel 1, and the Z and A axes are not used.

The circled button can be used to save, compile, download, and run the C program in one step. Note that once a C program has been executed to change configuration settings within the KMotion Board, the values in the KMotion Executive screens may be different from the current settings within the board. To synchronize the screens with what is actually in the board the channel should be "Uploaded".

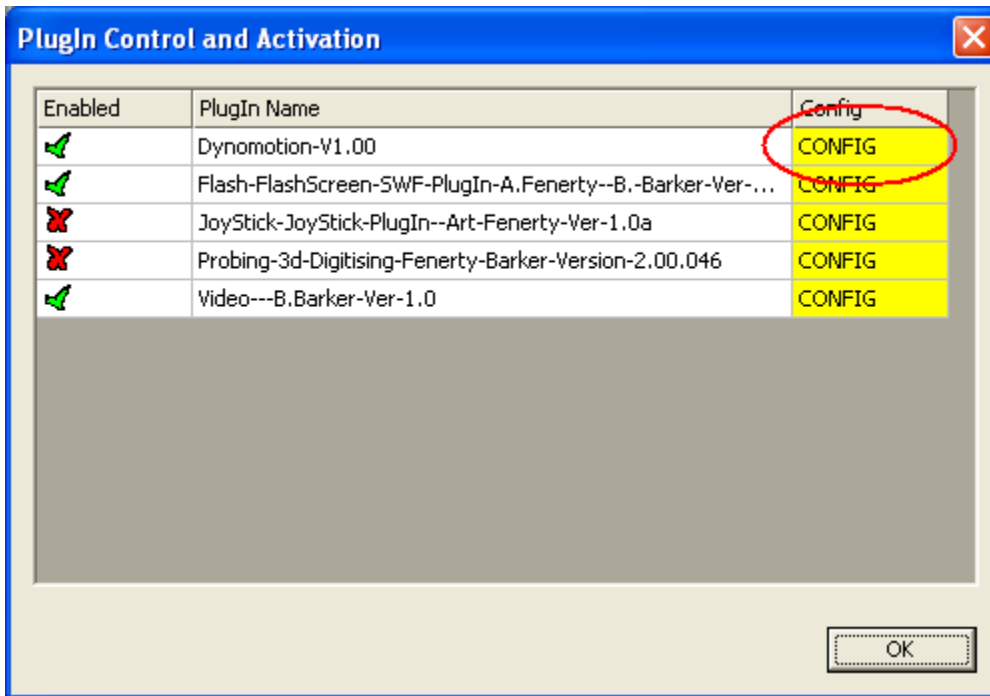
Additional initialization operations may also be added to the C Program. These might include brushless motor phase finding, homing, activating IO bits, etc..



After the Initialization program is finalized and working it should be saved in a known location so that Mach3 may be configured to execute it as described in the next step.

5. Within Mach3 - configure plugin (enter name of KMotion Init program)

We are now ready to Execute Mach3. The first time Mach3 is executed after adding the Dynomotion Plugin Mach3 will prompt the user which Motion Device should be used. The Dynomotion Plugin should be selected. After the Mach3 Application comes up use the Config|Config Plugins menu to bring up the Config Plugins dialog shown below.



Next click in the circled area to bring up the Dynomotion Plugin Configuration Screen.

Dynomotion Configuration V4.23C

RESET
 Initialize User Program: C:\KMotionSrc\C Programs\InitStepDir3Axis.c >>
☐ Auto run Initialize on Mach Startup ☒ Initialize Every Mach Reset

Spindle
 Spindle Speed User Program: C:\KMotionSrc\C Programs\SpindleMach3DAC.c >>
☐ Download Only Once ☐ Launch Only Once Thread: 2 Variables+0=msg, +1=dir, +2=speed 0
Threading
 Sensor Type: 1 0=none 1=encoder Encoder Axis: 0 Update Time: 0.2 secs
 Motion Filter Time Constant(Tau): 0.1 secs Count/Rev: 1024

Home
 Home User Program: C:\KMotionSrc\C Programs\HomeMach3.c >>
 Thread: 3 Variable flags: 5

Notify
 Custom Notify User Program: C:\KMotionSrc\C Programs\NotifyMach3.c >>
 Thread: 4 Variable Code: 6

Offline
 Mach3 OffLine drives: -1 bit no ☐ Neg True Polarity

Cycle Start
 Input Bit: -1 bit no ☐ Neg True Polarity

Windows Buffer ahead Time: 3 Seconds

Cancel OK

Specify the *KMotion Initialize User Program* that was created in step 4 above (you may browse to the file using the >> button).

Options to automatically run the program on Mach3 Startup or only once after Mach3 Startup may also be selected.

Mach3 generates Spindle Messages for On, Off, and Speed Changes. Specify a *KMotion Spindle Speed User Program* to handle and perform appropriate actions for the various Mach3 Spindle Messages. A default **SpindleMach3.c** program is included in the default installation that will simply print the messages and desired speeds requested by Mach3. Note that speed is a relative fraction of the max currently selected pulley's speed. Because KFlop/KMotion is likely to be controlling the motor speed regardless of the selected pulley, this is usually more appropriate. An appropriate User Program to perform the appropriate actions to actually drive the spindle should be created to replace the default program that simply prints the requested operation. There are included examples for Spindles controlled by a DAC output (SpindleMach3DAC.c) and for Spindles controlled by Step/Dir outputs (SpindleMach3Jog.c).

Spindle Speed measurement and Single point threading is also supported. A quadrature encoder is required on the spindle. Specify the Sensor Type as 1 to enable the Spindle measurement. Configure the Axis Channel that is configured to read the encoder (Note this is not the encoder channel, rather it is the axis that has the encoder channel configured as its input). Specify the Update time, Tau, and counts/rev. See [here](#) for more information.

Mach3 Home requests may be passed to a User Program to activate any desired Home Sequence. An example skeleton program which just prints the Mach3 home requests is included as **HomeMach3.c**. Another example for use with encoders is described [here](#).

Mach3 permits a general purpose mechanism to make custom calls to any Plugin that are passed through to a KFlop Custom Notify User Program. Placing a NotifyPlugins(10xxx) call in VB Script of either a Screen Button or Macro command may then trigger behavior in KMotion/KFlop. Message Codes from 10000 to 10999 will be sent to KFlop (all others are ignored). Plugin executions are automatically queued and executed in the sequence they were requested (the previous must complete before the next is launched). An example skeleton program which just prints the Mach3 notify requests is included as **NotifyMach3.c**. Another example for use with encoders is described [here](#).

An I/O bit may be defined that is activated when Mach3 goes into the offline state. Neg True Polarity may be checked if it is desirable for the bit to be driven low in the off-line state rather than high.

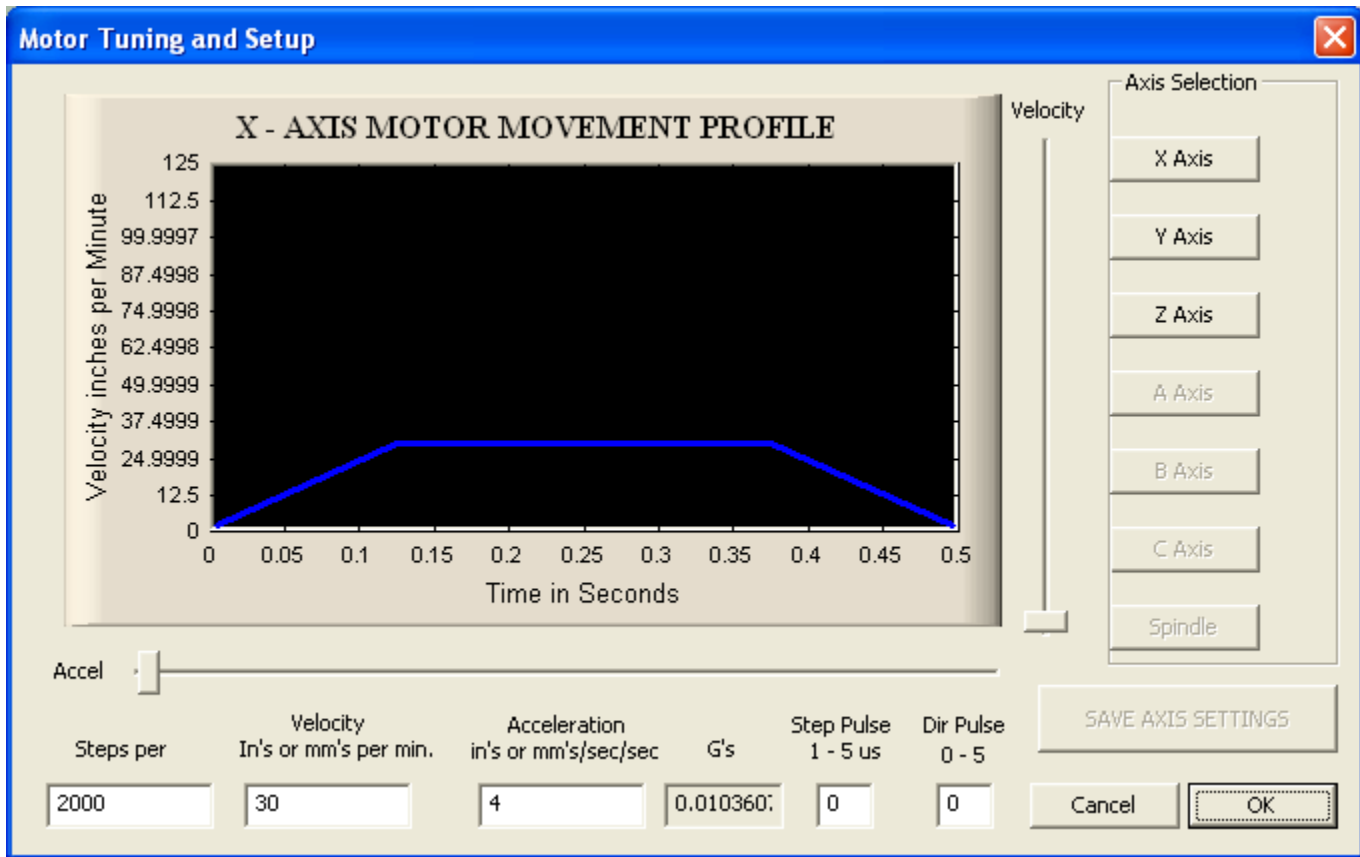
The *Windows Buffer ahead Time* may be changed to keep more or less buffering ahead in the KMotion Board. Using a value too small may cause the buffer to run empty and "starve" for data if Windows becomes non-responsive for longer than that amount of time. Using a value too large will cause feed rate changes and feed hold commands to be less responsive.

The plugin can be configured to *Automatically run Initialize on Mach Startup* if desired. **Caution:** This option should be left off if there is any potential danger with unexpected machine motion when launching Mach3.

After setting all parameters press OK to save the parameters into the Mach3 XML parameter file and return to the main Mach Screen. Pushing Mach3's Reset button will now execute the Initialize program.

6. Within Mach3 - configure motor tuning (set resolution, speeds, and acceleration)

Select **Config|Motor Tuning** to bring up the Motor Tuning Dialog. This screen allows the setting of Machine resolution, Max Velocity, and Max Acceleration for each axis in use. Note that the sliders are typically not very useful since the max step rates are much higher using a KMotion Motion Controller than with the original PC Generated Steps. It's usually best to just enter values directly into the cells.

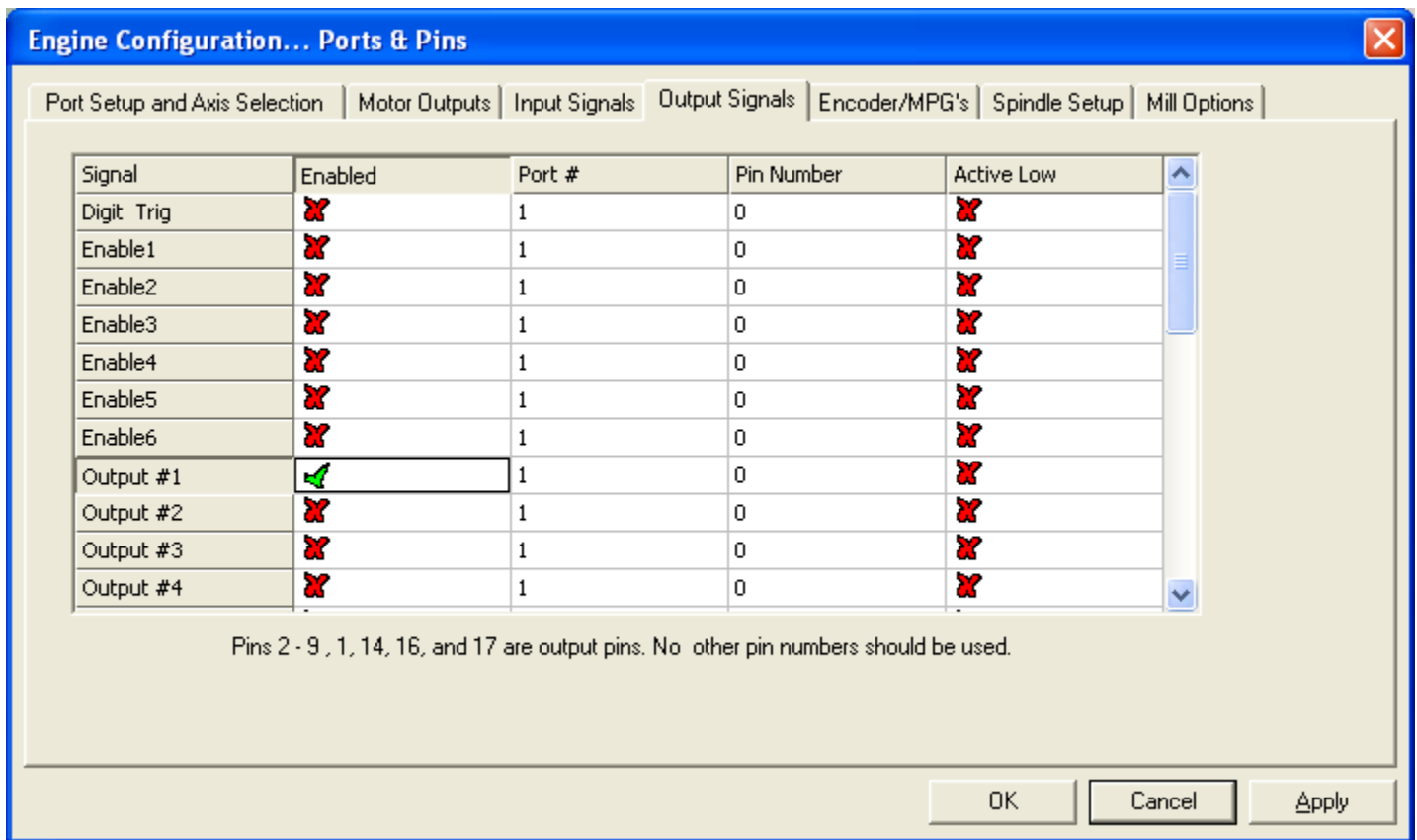


After the parameters have been entered and saved, Press OK to return to the main screen.

7. Within Mach3 - configure IO Bits.

Various M Codes may be used within Mach3 to activate IO bits on KMotion. Select the Menu **Config|Ports & Pins** to bring up the Dialog shown below. When using the KMotion Plugin, Pin Numbers now correspond to *KMotion IO Bit Numbers* rather than Parallel port Pins. Some of the terminology on the screen may be misleading as it was designed expecting to use a parallel port. For IO bit numbers 128 or larger, subtract 128 from the bit number and specify Port #2 instead of Port #1.

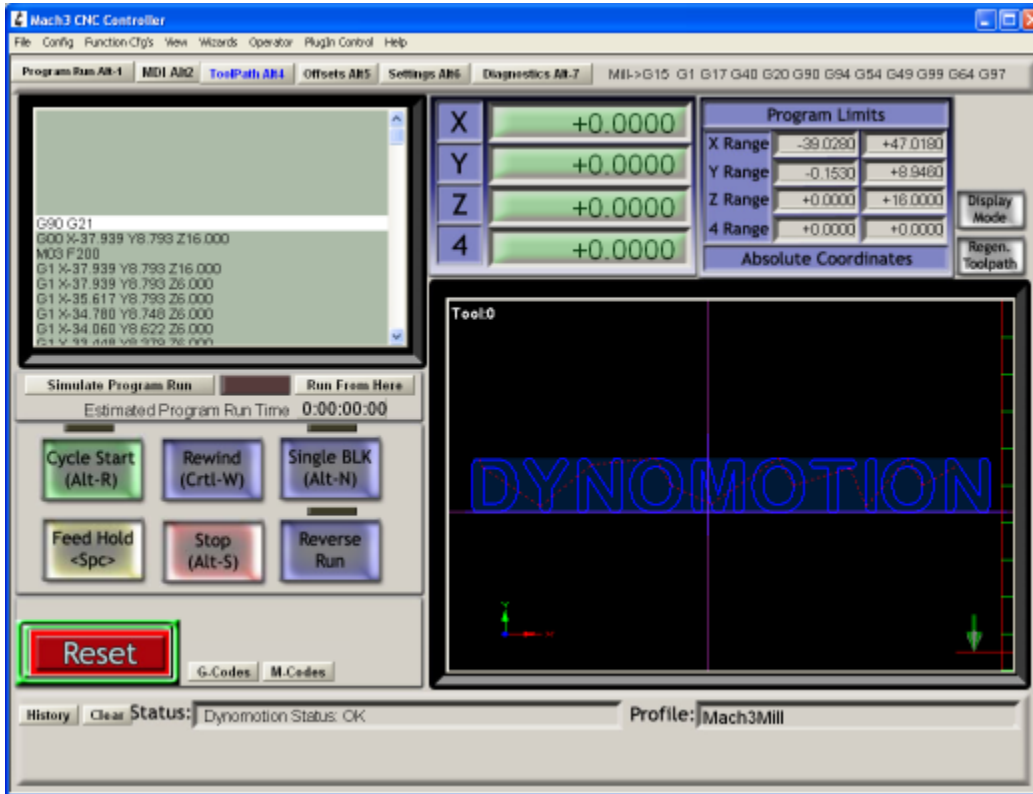
Note Mach3 typically defaults after an initial install with Output #1 enabled for spindle control on Pin0 (as shown below). Bit 0 is often used on a KMotion board as an Encoder input. Having Mach configure the IO bit as an output will cause the encoder to be inoperable. Disable the output if this is the case.



That's it! Reset should now properly initialize the machine. Jogging and G Code should now be functional.

Refer to the Mach3 documentation for more information on advanced features.

Here is how a GCode file appears on Mach3 Mill.



Here the same example GCode file cut (or more accurately burned) into a piece of wood using a Harbor Freight Mini-mill driven with KMotion and Mach3.

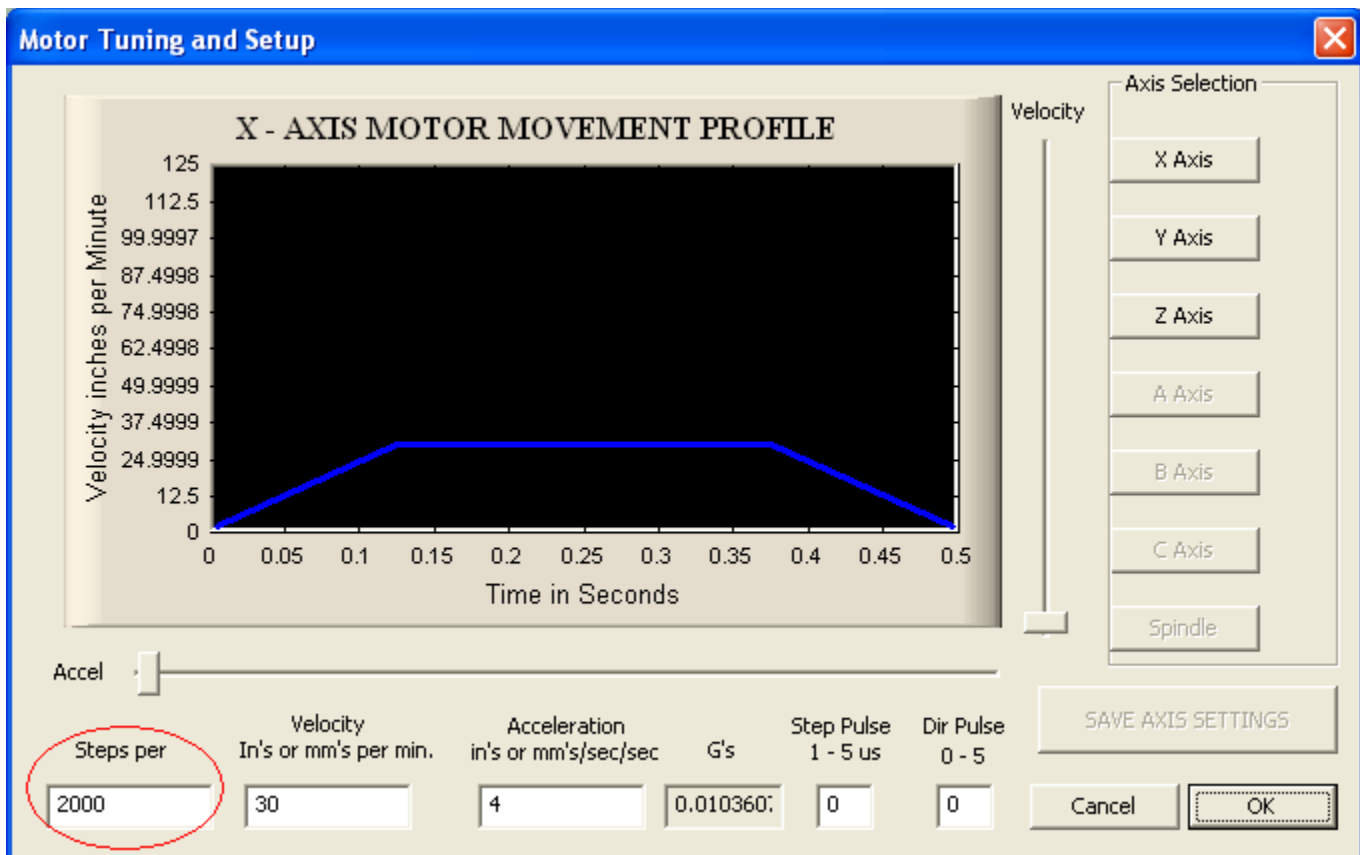


Mach3 Plugin - with Encoders

The following describes the use of linear glass scale encoders or rotary shaft encoders with Mach3. The setup process described apply for KFlop operating in open loop mode with encoders as well as [closed loop control](#).

KFlop/KMotion should first be wired and configured such that the encoders are functional and scaled (using the InputGain0 parameter) so that the encoder counts match the commanded position in units of μ steps. To verify that this is properly configured the KMotion.exe [Step Response Screen](#) may be used for verification. The Plot of Position Error should show small errors (typically $<200 \mu$ Steps) for a Move Plot if properly configured.

Since the encoder position is already scaled within KFlop/KMotion to match the μ steps/Unit scale of the motor, the Mach3 Encoder resolution should be set to the same value as the Motor "Tuning" as shown below. The Encoder/MPG screen is opened using the Config|Ports and Pins Menu. The Port and Pin definitions are not relevant when using KMotion/KFlop and should be set to some unused Port.



Engine Configuration... Ports & Pins

Port Setup and Axis Selection | Motor Outputs | Input Signals | Output Signals | Encoder/MPG's | Spindle Setup | Mill Options

Signal	Enabled	A -Port #	A -Pin #	B -Port #	B -Pin #	Counts/Unit	Velocity
Encoder1		0	0	0	0	2000	100.000000
Encoder2		0	0	0	0	2000	100.000000
Encoder3		0	0	0	0	2000	100.000000
Encoder4		0	0	0	0	1.000000	100.000000
MPG #1		0	0	0	0	1.000000	100.000000
MPG #2		0	0	0	0	1.000000	100.000000
MPG #3		0	0	0	0	1.000000	100.000000

OK Cancel Apply

Zero Buttons

Mach3 "Zeros" a DRO by adjusting the currently selected work offset such that the DRO will read zero. Since the glass scales are the best reference, the commanded position is adjusted to match the encoder position, before Mach3 is told to compute the new work offset.



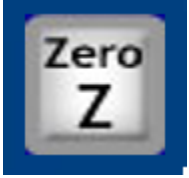
NotifyPlugins(10100) 'tell KFlop to set command to encoder
 Sleep 300 'make sure mach updates
 DoOEMButton (1008) 'calculate work offset



```

NotifyPlugins(10101)    'tell    KFlop    to    set    command    to    encoder
Sleep                  300      'make    sure    mach    updates
DoOEMButton (1009) 'calculate work offset

```



```

NotifyPlugins(10102)    'tell    KFlop    to    set    command    to    encoder
Sleep                  300      'make    sure    mach    updates
DoOEMButton (1010) 'calculate work offset

```

In the **Config|Config Plugins|Dynomotion** set an appropriate KFlop User Program that will process the NotifyPlugin Message Codes to set KFlops internal Commanded Destination to the Current Encoder Positions. Typical program for 3 axes shown below. Note that the Message code is defined to be passed to the KFlop User Program via `persist.UserData[6]`

Notify

Custom Notify User Program >>

Thread Variable Code

Example File: <Install Dir>\C Programs\NotifyZeroEncoderMach3.c

```

#include "KMotionDef.h"

//Plugin calls for Mach3 NotifyPlugins Commands
#define X 0
#define Y 1
#define Z 2

main()
{
    int msg = persist.UserData[6];  // Mach3 notify Message 10000-10999

    printf("Mach3 Notify Call, Message = %d\n",msg);

    if (msg==10100)
    {
        // adjust the commanded position to match the glass scale encoder
        DisableAxis(X);
        EnableAxisDest(X,chan[X].Position);
    }
    if (msg==10101)
    {
        // adjust the commanded position to match the glass scale encoder
        DisableAxis(Y);
        EnableAxisDest(Y,chan[Y].Position);
    }
    if (msg==10102)
    {
        // adjust the commanded position to match the glass scale encoder
        DisableAxis(Z);
        EnableAxisDest(Z,chan[Z].Position);
    }
    if (msg==10500)
    {
        if (CS0_StoppingState == 0)
            StopCoordinatedMotion();
        else
            ResumeCoordinatedMotion();
    }
    if (msg==10300)
    {
        // User wants to disable Z (switch to OL)
        DisableAxis(Z);
        chan[Z].OutputMode = STEP_DIR_MODE;

        EnableAxisDest(Z,chan[Z].Position);
    }
}

```

```

if (msg==10301)
{
    // User wants to enable Z (switch to CL)
    DisableAxis(Z);
    chan[Z].OutputMode = CL_STEP_DIR_MODE;

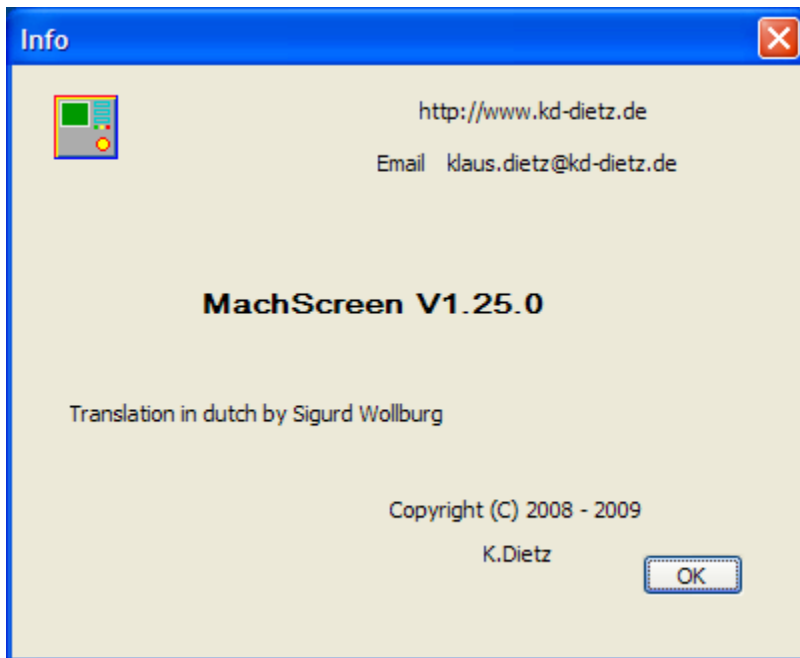
    EnableAxisDest(Z,chan[Z].Position);
}
}

```

REF Buttons

Mach3 REF buttons are used to set the initial Machine coordinates either by simply Zeroing them or performing a home operation into a switch.

The REF X, REF Y, REF Z etc... buttons may require editing using a screen editor. We recommend the one written by Klaus Dietz.



The Ref buttons should be edited to perform the standard Mach3 Ref operations. See the settings selected for the Ref buttons shown below when using Klaus' free Mach Screen Editor. The standard Ref operations for Mach3 is to request the Plugin to perform the Home Operation (actually labeled purge in the plugin).



Button

Description	Value
Global	No
Function	Ref X
OEM-Code	1022
Hotkey	
Execute code	
Text on ctrl	X Ref
Locked for mouse	No

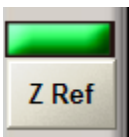
< [Progress Bar] >



Button

Description	Value
Global	No
Function	Ref Y
OEM-Code	1023
Hotkey	
Execute code	
Text on ctrl	Y Ref
Locked for mouse	No

< [Progress Bar] >



Button

Description	Value
Global	No
Function	Ref Z
OEM-Code	1024
Hotkey	
Execute code	
Text on ctrl	Z Ref
Locked for mouse	No

< [Progress Bar] >

The Dynomotion Plugin passes these Home requests to KFlop to handle with a Home User Program. In the **Config|Config Plugins|Dynomotion** set an appropriate KFlop User Home Program. A **flag** variable is also passed to tell which axis is to be homed (Note that the flag is defined to be passed to the KFlop User Program via `persist.UserData[5]`). In the case with encoders, both the Encoder Position and the Commanded Destination should be zeroed. Prior to Zeroing if any homing motion (to a switch for example - See: SimpleHome3Axis.c in the C Programs directory) may also be added into the program.

Home

Home User Program >>

Thread Variable flags

Example File: <Install Dir>\C Programs\HomeEncoderMach3.c

```

#include "KMotionDef.h"

//Plugin calls for Mach3 Home (actually Purge) Commands
//Called from Mach3 "REF" command
//in this case just Zero the measured position (encoder)
//and set the commanded destination to zero

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

main()
{
    int flags = persist.UserData[5]; // Mach3 flags bit0=X, bit1=Y,
    Bit2=Z, etc...

    printf("Mach3 Home Call, flags = %d\n",flags);

    if (flags & 1)
    {
        DisableAxis(X);
        Zero(X);
        EnableAxisDest(X,0.0);
    }
    if (flags & 2)
    {
        DisableAxis(Y);
        Zero(Y);
        EnableAxisDest(Y,0.0);
    }
    if (flags & 4)
    {
        DisableAxis(Z);
        Zero(Z);
        EnableAxisDest(Z,0.0);

        Jog(Z,10000); // start moving
        while (ReadBit(15)) ; // wait for switch (input #15) to change
        Jog(Z,0); // StopMotion

        while(!CheckDone(Z)) ;

        Jog(Z,-3000); // start moving
        while (!ReadBit(15)) ; // wait for switch (input #15) to change
        Jog(Z,0); // StopMotion
    }
}

```



```
    while(!CheckDone(Z)) ;  
    Delay_sec(.25);  
  
    DisableAxis(Z);  
    Zero(Z);  
    EnableAxisDest(Z,0.0);  
}  
}
```

Mach3 Plugin - Probe Setup

The following describes the use of DynoMotion Probing with Mach3. Probing in Mach3 is selected by using the G31 code. Such as:

G31 Z-4 F40

This example command line would cause a motion from the current position to a absolute position of $z = -4$ at a feed rate of 40 units/minute and stop as soon as the probe input becomes active. When the probe input becomes active the current positions of all axes will be captured and the motion will decelerate to a stop in a controlled manner. Mach3 variables 2000-2005 will be filled with the captured position.

The following sequence might be used to perform a probe and then perform a move relative to the captured Z position.

G31 Z-4.0 F40 (Probe in Z)

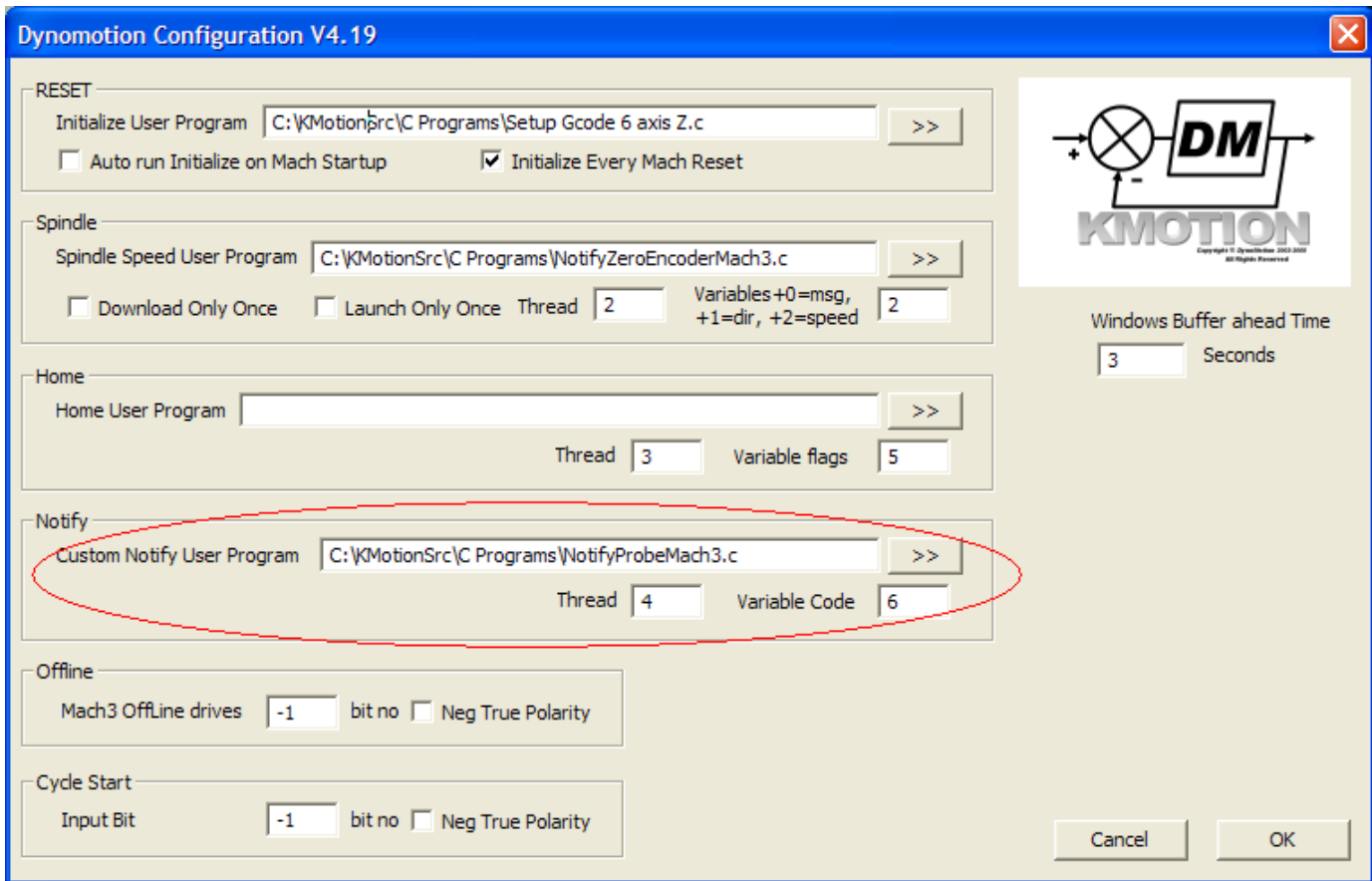
G1 Z #2002 (move back to trip point)

M30

Note: If the Probe input is already active at the beginning of the probe, or the motion completes without the Probe input ever becoming active, an error will be displayed and G Code execution will stop.

Required KMotion/KFLOP User Program

In order to perform Probing, the the Notify User Program must be configured to properly handle message 20000 and selected in the Mach3 Plugin configuration.



Add the message handler shown below to the Notify program for your system. During probing the DynoMotion Plugin sends a Notification 20000 message to the configured KMotion/KFlop Notify User Program.

The message handler must:

- #1 set status (user var 62) to zero
- #2 wait for the probe to become active
- #3 sample the current positions of the defined axes (as doubles into 50-61)
- #4 set status to 1 or 2 depending on if the probe was ever inactive
- #5 feedhold the system
- #6 exit

In most cases the only modification required will be the defined bit number and active state. However any number of other techniques might be used such as monitoring analog inputs, or capturing other variables such as encoder positions.

Excerpt from Example File: <Install Dir>\C Programs\NotifyProbeMach3.c

```

// handles probing
//
// flag is 0 - while watching for probe hit
// flag is 1 - if probe was already set from start
// flag is 2 - after successful probe hit
//
// returns the captured results in User Variables
// X - 50+51
// Y - 52+53
// Z - 54+55
// A - 56+57
// B - 58+59
// C - 60+61
// status result 62

#define PROBE_BIT 0
#define PROBE_ACTIVE_STATE 1

if (msg==20000)
{
    double *d = (double *)&persist.UserData[50];
    int flag=1;

    persist.UserData[62]=0;

    while (ReadBit(PROBE_BIT)!=PROBE_ACTIVE_STATE)
    {
        flag=2;
        WaitNextTimeSlice();
    }

    if (CS0_axis_x>=0) d[0]=chan[CS0_axis_x].Dest;
    if (CS0_axis_y>=0) d[1]=chan[CS0_axis_y].Dest;
    if (CS0_axis_z>=0) d[2]=chan[CS0_axis_z].Dest;
    if (CS0_axis_a>=0) d[3]=chan[CS0_axis_a].Dest;
    if (CS0_axis_b>=0) d[4]=chan[CS0_axis_b].Dest;
    if (CS0_axis_c>=0) d[5]=chan[CS0_axis_c].Dest;

    persist.UserData[56]=flag;
    StopCoordinatedMotion();
}

```

Mach3 Plugin - Passing DROs

Mechanism for transferring values back and forth between Mach3 and KFLOP

Mach3 User DROs 1 to 50 map to KFlop UserData 0 to 99 (2 words each double)

To Read from KFLOP to Mach3 use NotifyPlugins codes 18001 to 18050

To Write from Mach3 to KFLOP use NotifyPlugins codes 19001 to 19050

Example MACH3 SIDE

```
SetOEMDRO(1007,123.456) 'Put a value in a Mach DRO
NotifyPlugins(19007) 'Send it to KFLOP
Sleep(3000) 'Wait for KFLOP to modify and copy it
NotifyPlugins(18008) 'Read the result from KFLOP
x=GetOEMDRO(1008) 'Check the value passed back
```

Example KFLOP SIDE

```
#include "KMotionDef.h"
#define DROIN 7
#define DROOUT 8

main()
{
    double *pin = (double *)&persist.UserData[(DROIN -1)*2];
    double *pout = (double *)&persist.UserData[(DROOUT-1)*2];
    for(;;)
    {
        Delay_sec(2);
        *pout = *pin + 999;
        printf("DROIN %d = %f DROOUT %d=%f\n", DROIN, *pin, DROOUT, *pout);
    }
}
```

Mach3 Plugin - Rigid Tapping

To perform Rigid Tapping from Mach3 the tapping parameters are set into GCode Variables, then a Macro (M84) is called which downloads the parameters to KFLOP, notifies KFlop to perform the Tap Cycle, then waits until KFLOP sets a variable indicating the operation has completed.

A requirement for Rigid Tapping is that the Spindle has encoder feedback and is possible to move in a fairly controlled manner. The Z axis motion is "geared" to the measured Spindle Encoder Position throughout the cycle.

There are three parts to the process: The GCode, the M84 Macro, and the KFlop User Program. Examples can be found [here](#).

Example Rigid Tap Call From GCode

We use a Macro M84 as Mach3 uses the normal G84 Tap cycle for a floating tapholder technique and doesn't currently support a rigid tap GCode.

Note that the forward cutting rate (RPM) and the retraction rate (RPM) can be defined separately. A cyclic forward/retract motion can be specified to cut the thread to the total depth. If a simple single motion is desired, set the Z depth forward motion to the Z depth Total.

G0X0Y0Z5

(Call a Rigid Tap Sequence)

#10=20 (TPI - Threads per inch)

#11=700 (Forward Cutting RPM)

#12=1000 (Retract RPM)

#13=0.75 (Z depth Total inches)

#14=0.2 (Z depth Forward per Motion)

#15=0.05 (Z depth Retract per Motion)

M84

G0X4Y0Z5

(Call a Rigid Tap Sequence)

#10=20 (TPI - Threads per inch)

#11=700 (Forward Cutting RPM)

#12=1000 (Retract RPM)

#13=0.75 (Z depth Total inches)

#14=0.2 (Z depth Forward per Motion)

#15=0.05 (Z depth Retract per Motion)

M84

M2

Mach3 M84 Macro

This macro moves the GCode Tapping Variables to Mach3 User DROs, downloads then to KFLOP UserData variables, Triggers KFLOP to perform the Tap Cycle, then waits until KFLOP sets a User Data Variable indicating the cycle is complete.

```
'Macro for Rigid Tappin with Dynomotion KFLOP
```

```
,
' pass variables to KFLOP
,
' Var DRO KFLOP UserVar Description
' #10 1010 18 19 TPI - Threads per inch
' #11 1011 20 21 Forward Cutting RPM
' #12 1012 22 23 Retract RPM
' #13 1013 24 25 Z depth Total inches
' #14 1014 26 27 Z depth Forward per Motion
' #15 1015 28 29 Z depth Retract per Motion
' 1016 30 31 Set by KFLOP when complete
```

```
'Move the GCode Vars into DROS and send them to KFLOP User Vars
```

```
For i=0 To 5
Call SetUserDRO(1010+i, GetVar(10+i))
NotifyPlugins(19010+i)
Next i
```

```
Call SetUserDRO(1016, 0) 'clear the complete flag
NotifyPlugins(19010+6)
```

```
NotifyPlugins(10084) 'do the TAP!!
```

```
While GetUserDRO(1016)=0
Sleep(50)
NotifyPlugins(18016) 'upload the complete flag
Wend
```

KFLOP Notify User C Program that performs the Rigid Tap Cycle

The C Program that performs the Rigid Tap Cycle. This assumes that the Spindle axis can be controlled like a Servo Axis within KFLOP (although it is not defined as an axis within Mach3). The defines must be set per your specific system. A low pass filter is used to smooth the commanded Z motion for the case where the Spindle Motion might be too Jerky for the Z Axis to follow without possibly stalling, it also smoothes the response that would be otherwise stepped because user programs only execute every other servo sample. A Tau of 0.001 performs as a low pass filter with a time constant of 1ms.

```

#include "KMotionDef.h"

//Plugin calls for Mach3 NotifyPlugins Commands

void Tap(void);

main()
{
    int msg = persist.UserData[6];  // Mach3 notify Message 10000-10999

    printf("Mach3 Notify Call, Message = %d\n",msg);

    if (msg==10084)
    {
        Tap();
    }
}

//  R I G I D   T A P P I N G

#define ZAXIS 7
#define SPINDLE_AXIS 6
#define Z_CNTS_PER_INCH -20000.0
#define CNTS_PER_REV (8192*14/16)
#define TAU 0.001

double SlaveGain,ToCut,TotalCut,Z0,S0;
void DoSlave(void);
void DoTap(double Dist, double Rate, double TPI);

void Tap(void)
{
    //  #10  1010      18 19          TPI
    //  #11  1011      20 21          Forward Cutting RPM
    //  #12  1012      22 23          Retract RPM
    //  #13  1013      24 25          Z depth Total inches
    //  #14  1014      26 27          Z depth Forward per Motion
    //  #15  1015      28 29          Z depth Retract per Motion
    //  #16  1015      30 31          Complete Flag

    double TPI                = *(double *)&persist.UserData[18];
    double CutRPM              = *(double *)&persist.UserData[20];
    double RetractRPM          = *(double *)&persist.UserData[22];
    double ZDist               = *(double *)&persist.UserData[24];
    double ZForward            = *(double *)&persist.UserData[26];
    double ZReverse            = *(double *)&persist.UserData[28];

    double FeedRate            = CutRPM/(TPI*60);

```



```

double RetractRate  = RetractRPM/(TPI*60.0);

printf("TPI = %f\n",TPI);
printf("FeedRate = %f\n",FeedRate);
printf("RetractRate = %f\n",RetractRate);
printf("ZDist = %f\n",ZDist);
printf("ZForward= %f\n",ZForward);
printf("ZReverse = %f\n",ZReverse);

// Slave the Z Axis to the Spindle
SlaveGain = Z_CNTS_PER_INCH/(CNTS_PER_REV * TPI);
Z0 = chan[ZAXIS].Dest;
S0 = chan[SPINDLE_AXIS].Dest;

// in case there is significant spindle position error move there
first
Move(ZAXIS,(chan[SPINDLE_AXIS].Position-S0)*SlaveGain+Z0);
while (!CheckDone(ZAXIS)) ;

TotalCut=0.0;
while (TotalCut < ZDist)
{
    if (TotalCut + ZForward > ZDist) // last feed
    {
        // yes, do any remaining
        DoTap(ZDist-TotalCut, FeedRate, TPI);
        // retract fully
        DoTap(-ZDist, RetractRate, TPI);
        TotalCut=ZDist;
    }
    else
    {
        // no, just cut a bit
        DoTap(ZForward, FeedRate, TPI);
        DoTap(-ZReverse, RetractRate, TPI);
        TotalCut+=ZForward-ZReverse;
    }
}

Delay_sec(1.0);
Move(ZAXIS,Z0); // move back to where we started
while (!CheckDone(ZAXIS)) ;

*(double *)&persist.UserData[30]=1.0; // set flag that we are
complete
printf("Tap Complete\n");
}

void DoTap(double Dist, double Rate, double TPI)

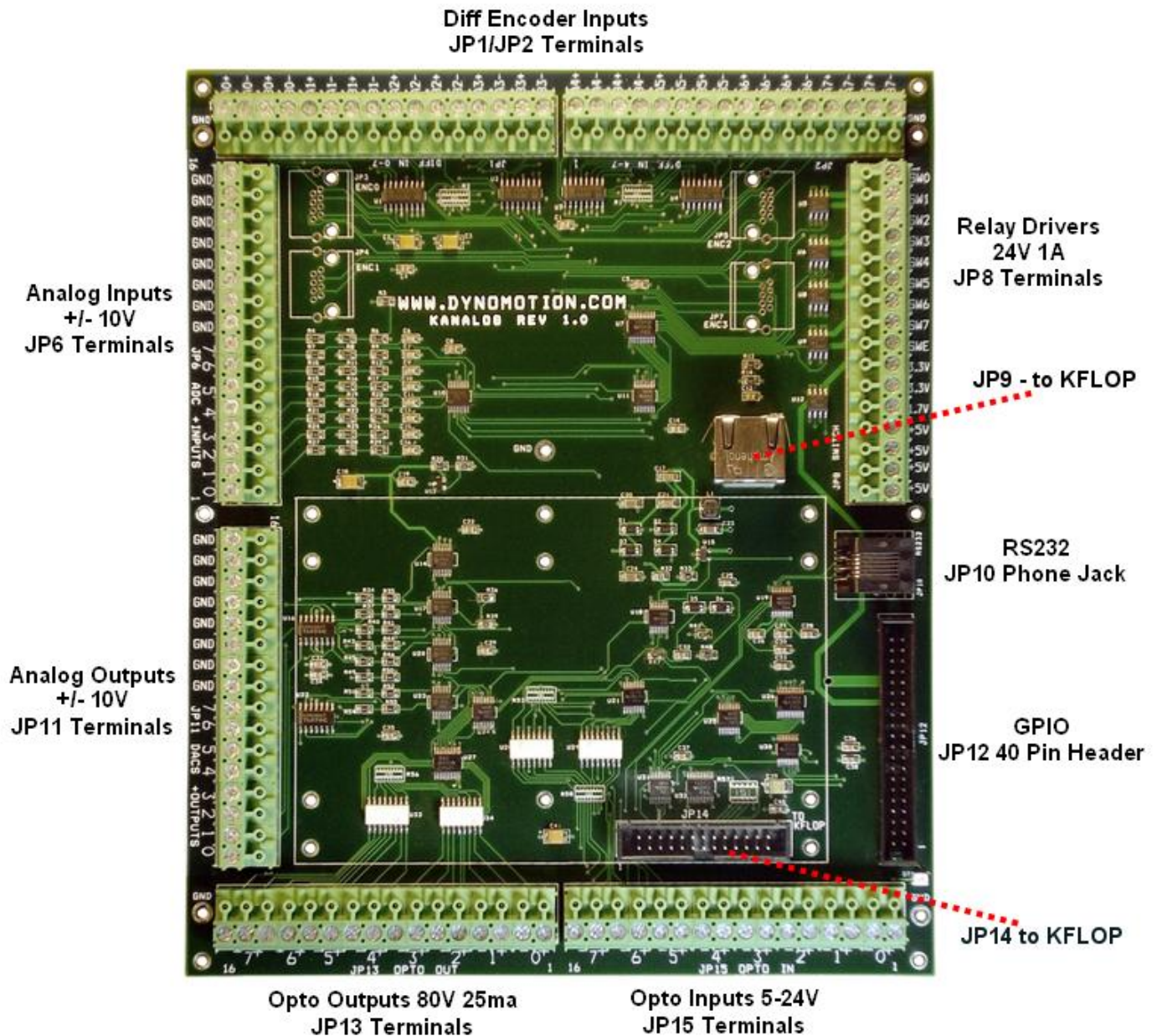
```

```
{  
    // Tap down  
    MoveRelAtVel (SPINDLE_AXIS, Dist*TPI*CNTS_PER_REV,  
Rate*TPI*CNTS_PER_REV);  
  
    while (!CheckDone (SPINDLE_AXIS))  
        DoSlave ();  
}  
  
void DoSlave (void)  
{  
    MoveExp (ZAXIS, (chan[SPINDLE_AXIS].Dest-S0)*SlaveGain+Z0, TAU);  
    WaitNextTimeSlice ();  
}
```

Kanalog 1.0 Hardware

Function	Parameter	Specification
Analog	DAC Outputs ADC Inputs	(8) +/- 10V @ 10ma, recommended load >2Kohm 12 bit - update rate 11.1KHz all 8 channels (8) +/- 10V 100Kohm input Impedance 12 bit - update rate 11.1KHz all 8 channels
Optos	Opto Isolated Outputs Opto Isolated Inputs	(8) Max differential voltage 80V 25ma (8) 3-24V input, internal 10KOhm series resistor
Differential	Differential Receivers Input	(16) ANSI TIA/EIA-422-B, ANSI TIA/EIA-423-B +/-7V common mode range, 200mv sensitivity Internal 470ohm termination
Relay Drivers	N channel FET Switches	(8) 24V @ 1 Amp max Open Drain switches to ground
Digital I/O	GPIO - LVTTL	(8) Outputs Source/Sink 12ma (8) Inputs LVTTL, 3.3V 1µa max
±15V	On-board DC-DC Generator	2 Watts (70ma each supply)
RS232	Driver/Receiver	EIA/TIA-232-F
Terminal Strips	Screw Terminals	112 Pluggable Screw Terminals (7) x 16 pin 5mm pitch
Watchdog	Charge Pump	DSP communication enables (1) Relay Driver, FET Switch 24V @ 1A
Logic Supply	Voltage Typical Current	+5V ±10% 0.5 A
Environment	Operating Temperature Storage Temperature Humidity	0-40° C 0-40° C 20-90% Relative Humidity, non-condensing
Dimensions	Length Width Height	8.5 inches (216mm) 7.0 inches (178 mm) 0.75inches (19 mm)
Green	RoHS	Compliant

Kanalog - Connector Pinouts



JP1/JP2 Differential Inputs

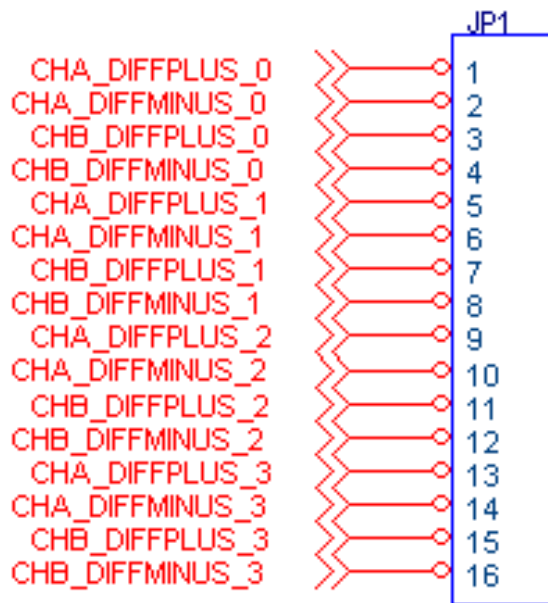
16 Differential Receiver Inputs are provided. Kanalog converts the differential signals to signal ended LVTTTL signals and passes them through to existing KFlop I/O pins.

JP1 converts 8 of the signals and connects them to KFlop I/O bits 0-7 (KFlop JP7 Pins 7-14) which are KFlop's 4 encoder A/B input channels.

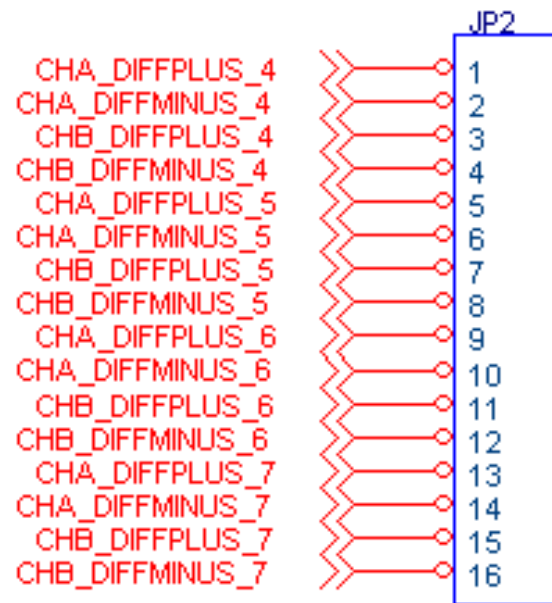
JP2 converts 8 of the signals and connects them to Kflop I/O bits 36-43 (Kflop JP5 Pins 1-8) which are General Purpose I/O pins.

Typically up to 4 encoder's A/B signals are connected to JP1 and any Z index inputs are connected to JP2

These differential inputs are moderately damped with an internal 470ohm resistor connected across the + to - inputs. If additional termination is required an external resistor may be connected.



Terminal DIFF IN 0-3

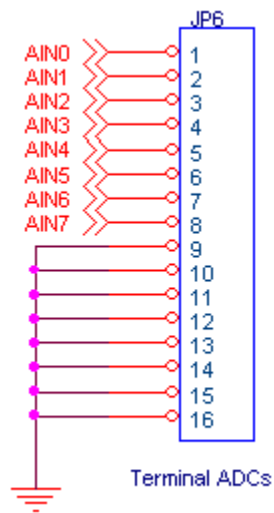


Terminal DIFF IN 4-7

JP6 - Analog Inputs +/- 10V

(8) +/- 10V analog inputs are provided. These are mapped and referenced as ADC inputs 0-7 for Kflop configuration purposes. Input impedance is approximately 100K ohms.

(8) ground terminals are provided. Kanalog contains a single solid ground plane so these grounds may be used as any digital or analog ground connection.



JP8 - FET Switch Outputs (relay drivers) - watchdog - power outputs

(8) 24V @ 1Amp relay FET Switch Outputs are available on Kanalog mapped as Output Bits 152-159. When activated (checked on or with state "1") the FET Switches make a connection to ground. Normally a load, such as a relay coil is connected between some appropriate +supply and a Switch input. Therefore when the Switch makes a connection to ground, the load is energized.

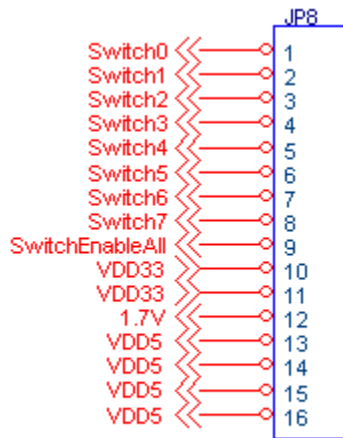
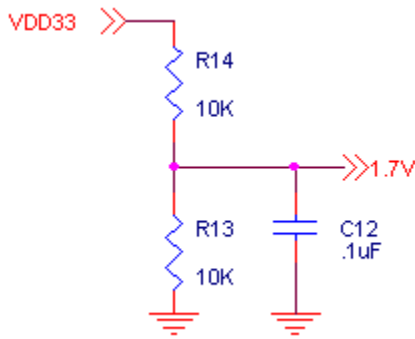
Caution! Inductive loads MUST have a reverse diode connected in parallel with the load to avoid a high voltage spark when the switch opens. Failure to allow a re-circulating current path on any inductive coil such as a relay, solenoid, or motor is likely to cause permanent damage.

One Enable or Watchdog FET Switch Output is also available (24V @ 1Amp). This switch conducts after Kflop boots, enables the +/-15V generator, and begins communicating with the board. It is recommended that this switch output is used as one of the conditions to enable main system power for motors and other devices.

(2) +3.3V outputs are available to power low current (<100ma) external circuitry.

One low current 1.7V bias current is available. See circuit below:

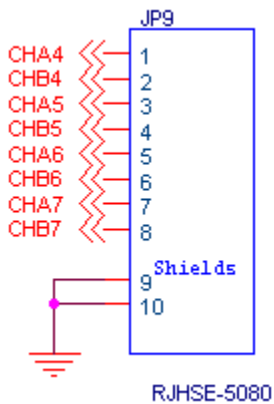
(4) +5V outputs are available to power low current (<100ma) external circuitry such as encoders. Normally +5V is applied to Kflop JR1 (4 pin white Molex connector) and passes through the ribbon connector to Kanalog. However it is also possible to feed +5V into the system via these terminals. If +5V power is fed into both Kflop and Kanalog the exact same +5V supply must be connected to both.



Terminal Relay Drivers/Power

JP9 - Differential Signals 8-15 to Kflop

The second (8) of the 16 Differential signals pass through to Kflop through this connector. If only the first 8 differential are used then this connector is not required and the 8 Kflop inputs may be used for some other purpose.

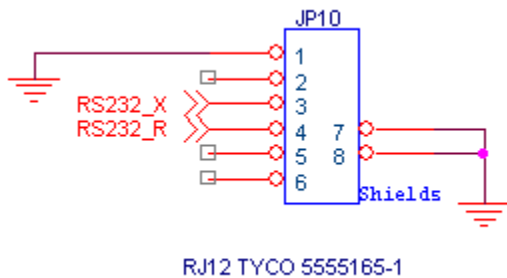


RJHSE-5080

JP10 - RS232

The JP10 6-pin phone connector provides 3-wire RS232 connectivity. JP10 Pin3 is used for transmit data from Kanalog. JP10 Pin 4 is used for receive data going into Kanalog. Receive data is converted to LVTTTL and routed to a Kflop I/O bit #44. Data from LVTTTL Kflop I/O bit #45 is passed through the RS232 driver and out the Transmit pin. This phone plug connector is designed to be compatible with [Automation Direct's](#) PLC line using a straight through phone cable.

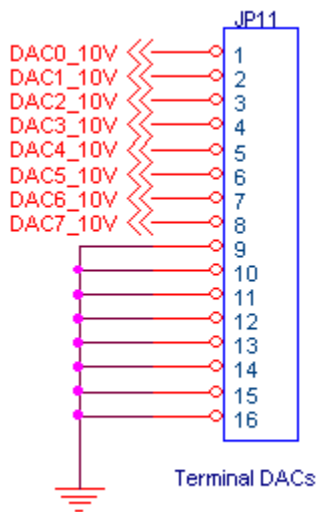
Note: Currently the RS232 capability of Kanalog is not supported in the Kflop firmware. Contact Dynomotion for current status if RS232 is required in your application.



JP11 - Analog Outputs +/- 10V

(8) +/- 10V analog outputs are provided. These are mapped and referenced as DAC outputs 0-7 for Kflop configuration purposes. Output impedance is approximately 2K ohms.

(8) ground terminals are provided. Kanalog contains a single solid ground plane so these grounds may be used as any digital or analog ground connection.



JP12 - General Purpose Inputs, Outputs, Low level analog in, Supplies

JP12 is a standard 40 pin IDC header connector.

(8) 3.3V LVTTL Inputs are provided (SDIN0 - SDIN7) mapped as Kanalog Input bits #128-135. Inputs are diode clamped to 3.3V.

Note: To connect 5V signals a 200ohm external series resistor is required.

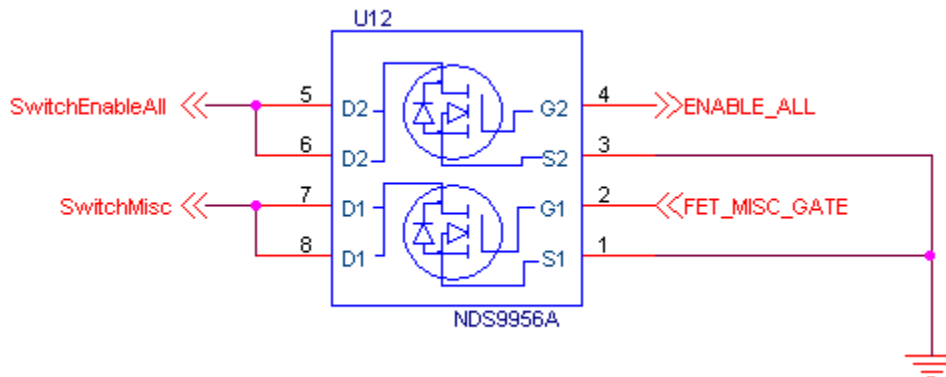
(8) 3.3V LVTTL Outputs are provided (OUT 0 - OUT_7) mapped as Kanalog Output bits #160-167.

ADC channels 0-3 have internal low voltage inputs exposed as signals IN0-IN3. Instead of driving the normal terminal inputs with a voltage range of +/- 10V. The INx pins may be driven with low voltage (0V - 3V) signals with an input impedance of ~10K ohms. This may allow higher resolution with low voltage signals. Caution should be used as these are low voltage unprotected signal inputs.

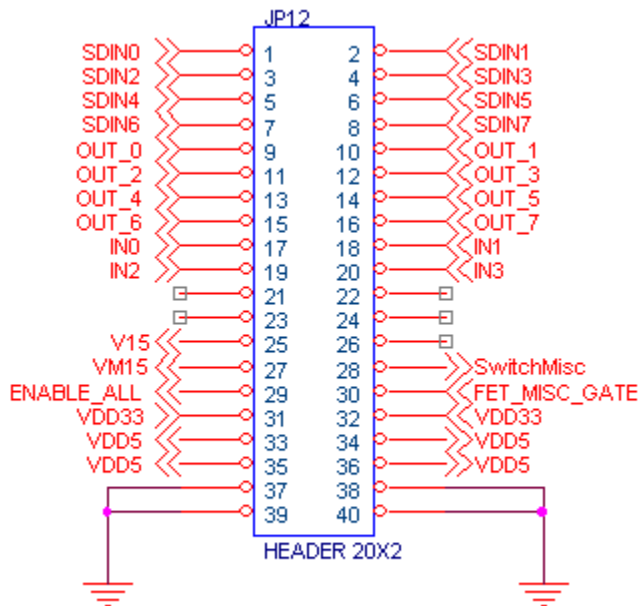
+/- 15V from the internal DC-DC generator is available on pins V15 (+15V) and VM15 (-15V). 70ma is available for external use for each supply.

ENABLE_ALL is the LVTTL equivalent of the SwitchEnableAll FET output if a logic level is desired instead of the FET SWITCH output.

One additional 24V 1Amp FET switch is available with an exposed gate. Drive signal FET_MISC_GATE to 3.3v to turn on the FET (SwitchMisc) output.



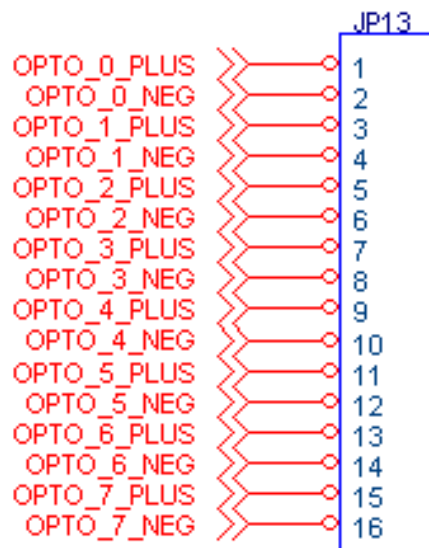
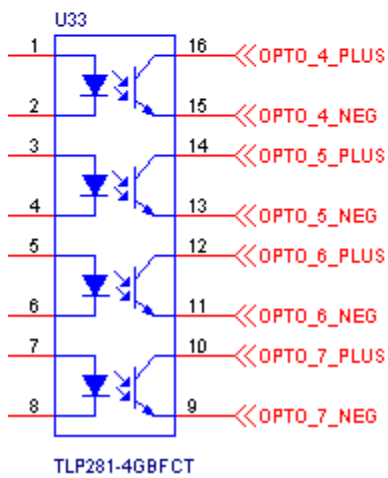
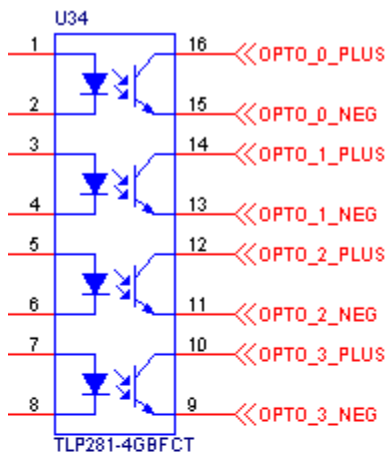
+3.3V and +5V are available on this connector. +3.3V is available as an output only (regulated down from the +5V). +5V is common to all other +5V signals and may be used as input or output.



JP13 - Opto Outputs

(8) totally isolated and independent optically isolated outputs are provided.

Opto output transistors are rated for a max voltage of 80V and will conduct up to 25ma of current.



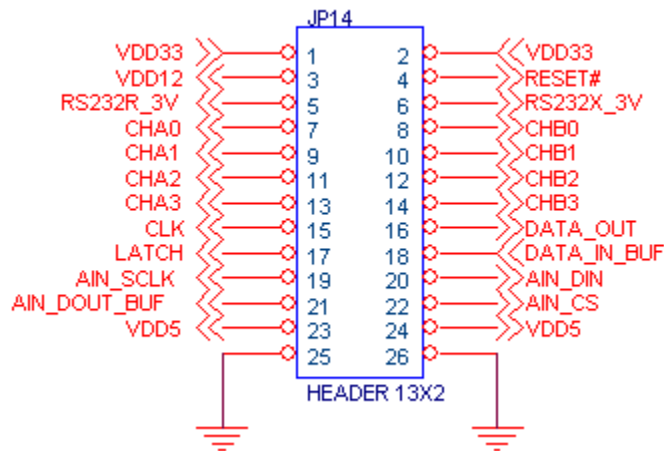
Terminal Opto Out

JP14 - Kflop

This is the main connection between Kflop and Kanalog and should consist of a short 26-pin one-one ribbon cable.

Analog and digital data passes through this cable in serial form. The first 8 differential signals pass to the Kflop in parallel form.

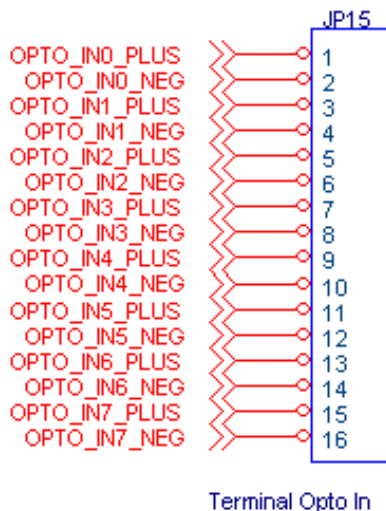
None of these signals should be used by the User.

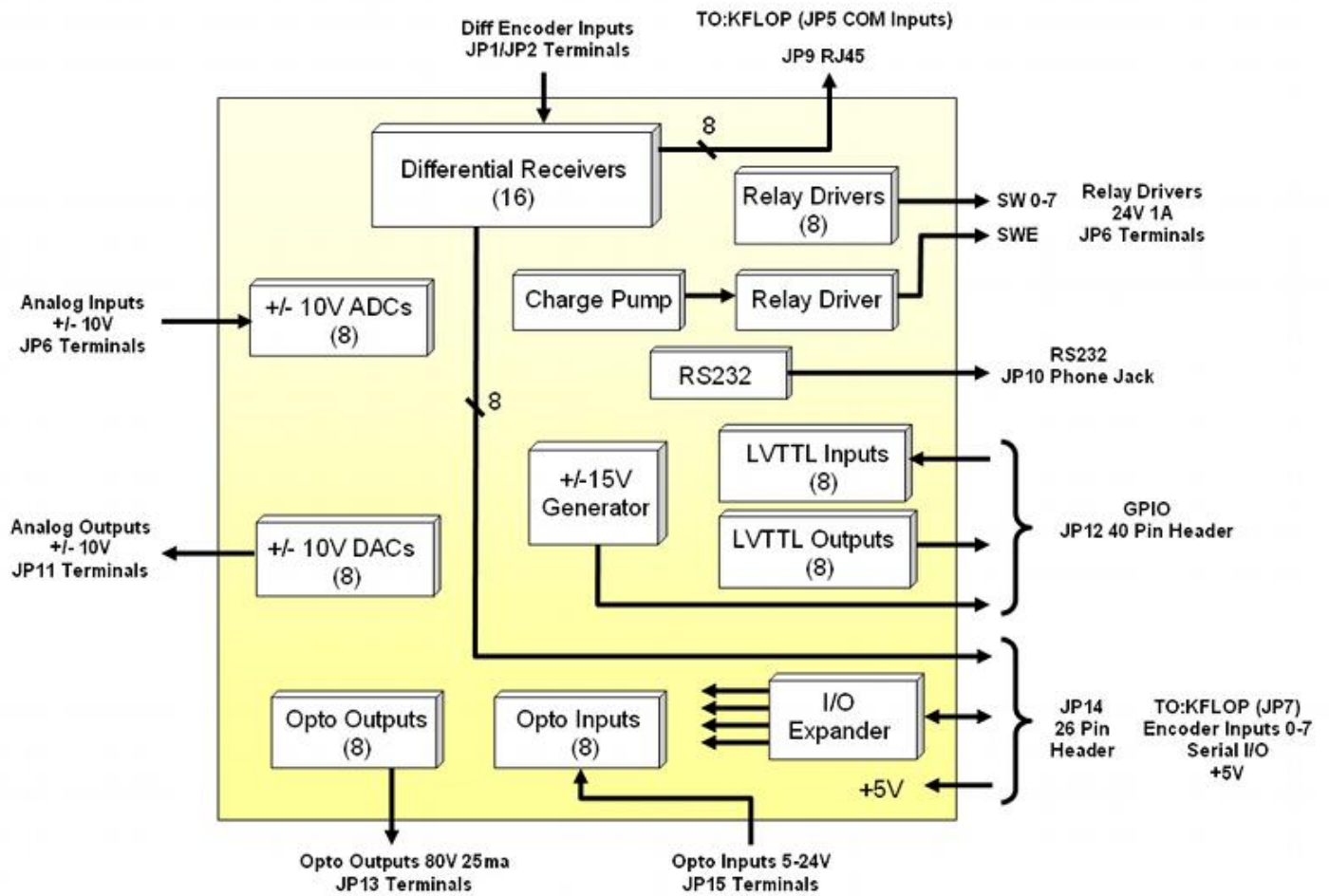


JP15 - Opto Inputs

(8) totally isolated and independent optically isolated inputs are provided.

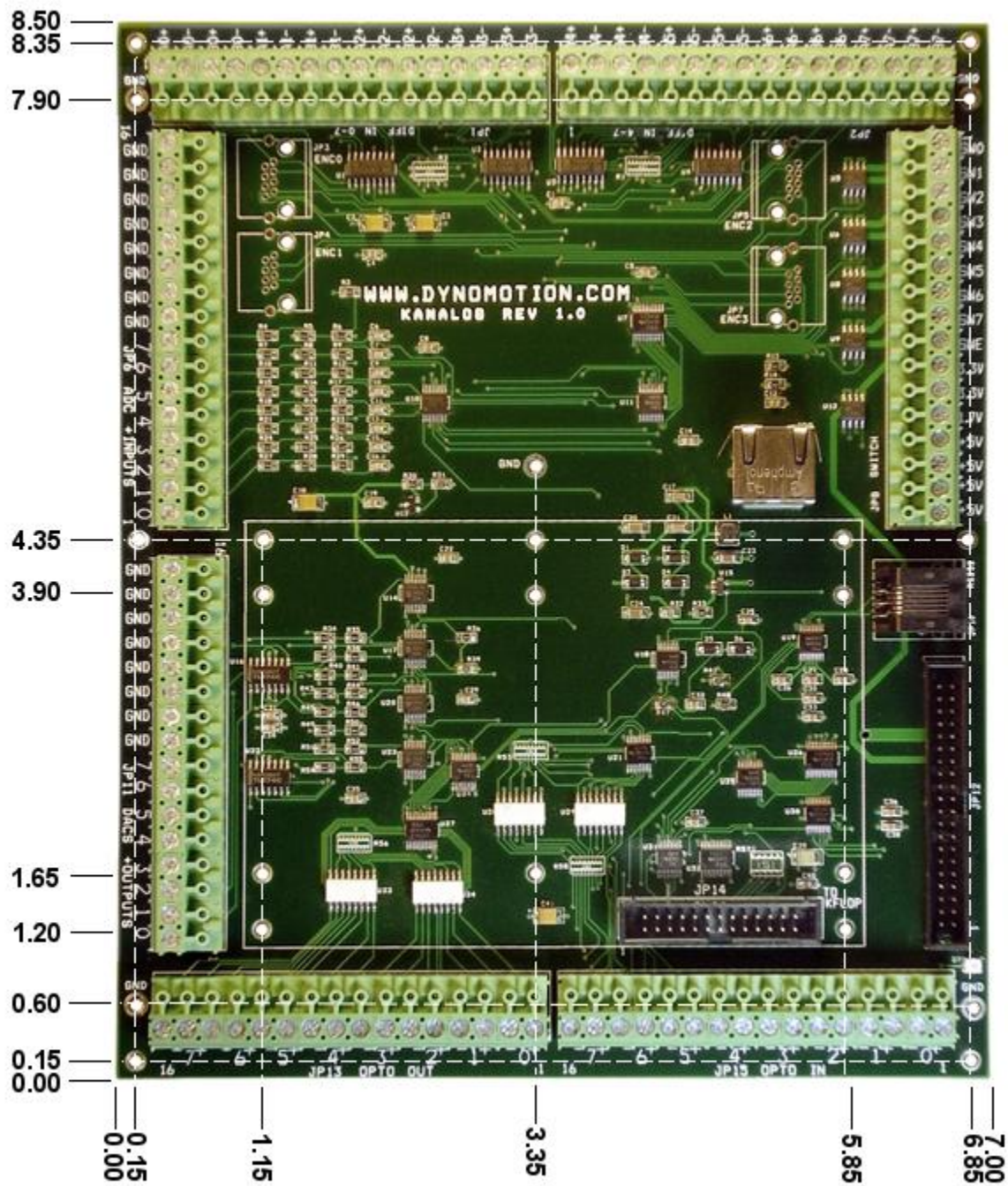
Input LEDs have a series resistance of 10Kohms and may be driven directly by any voltage from 5-24V. Max drain of 2.4ma when driven with 24V.



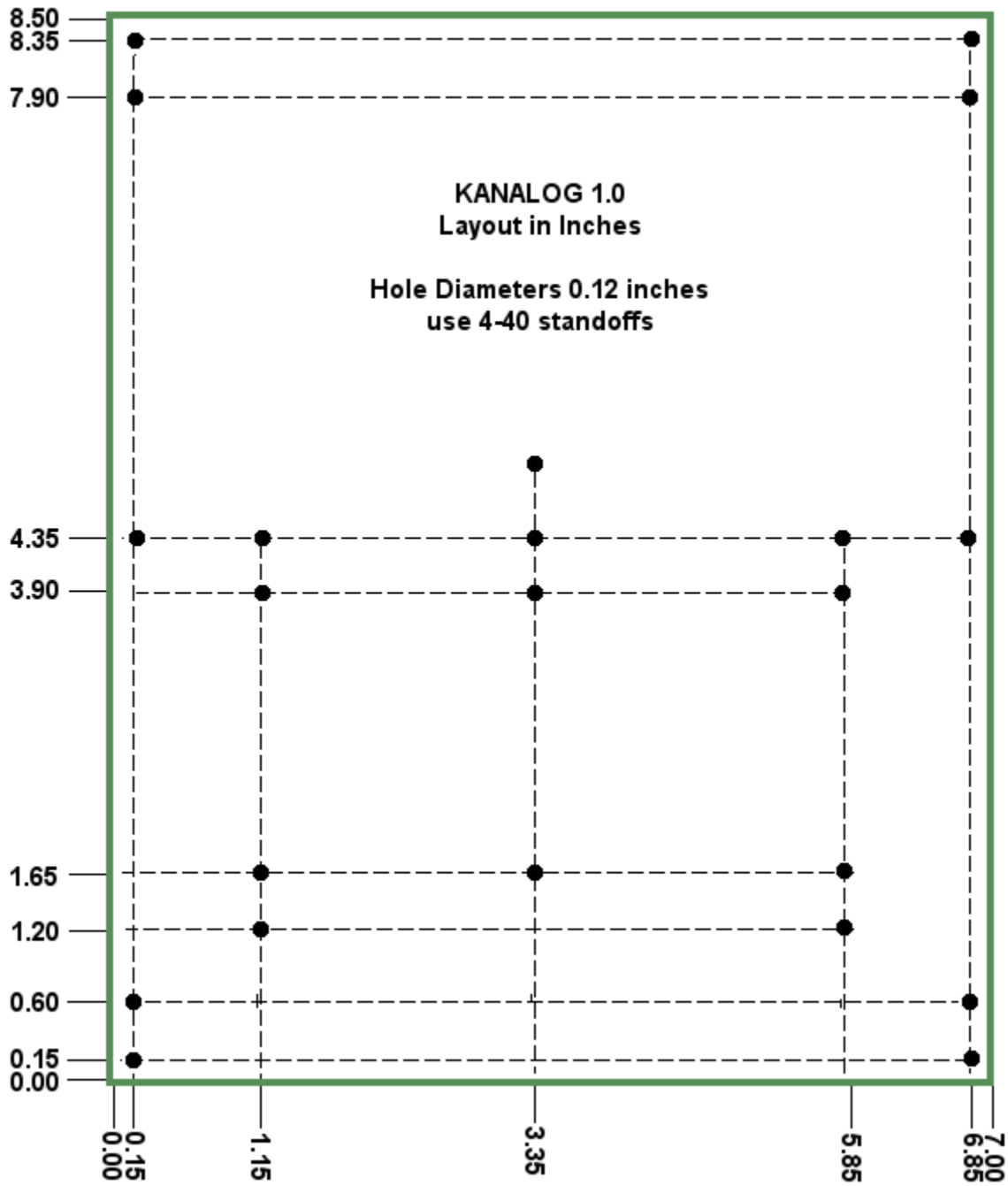


Kanalog Block Diagram

Kanalog Board Layout



[Show Photo](#)



Using Kanalog 1.0

Kanalog adds Analog and Digital I/O to Dynamotion's *KFLOP* motion controller. *Kanalog* provides 6 types of various I/O which is enough to completely drive many types of machine tools.

Standard +/-10V Analog outputs may be used to drive Analog Motor Drives. The Analog outputs are true 12 bit DACs, *not* filtered PWMs that have slow response and ripple. All (8) Analog inputs and (8) Analog outputs are all updated every Servo Sample time of 90us (11KHz).

Relay Drivers, Opto Inputs, Opto Outputs, Differential Encoder Inputs, LVTTTL inputs and outputs are all included along with 112 screw terminals.

The photo below shows *Kanalog* with *KFlop* mounted and two cables that connect the two boards.

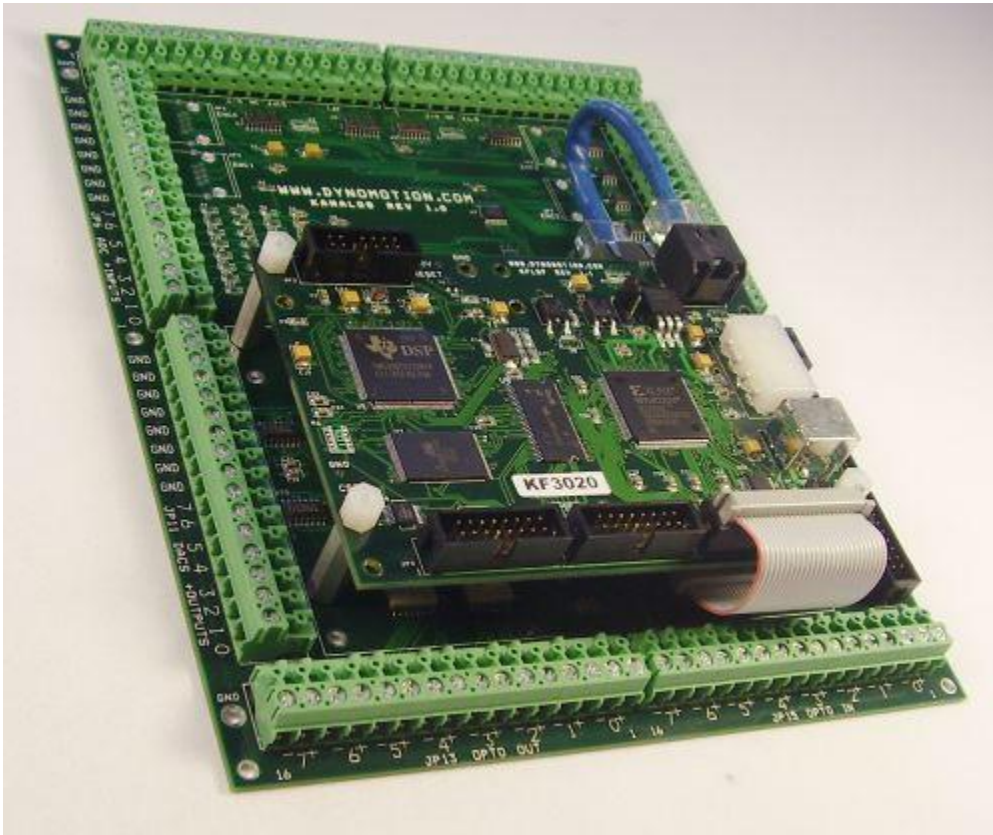
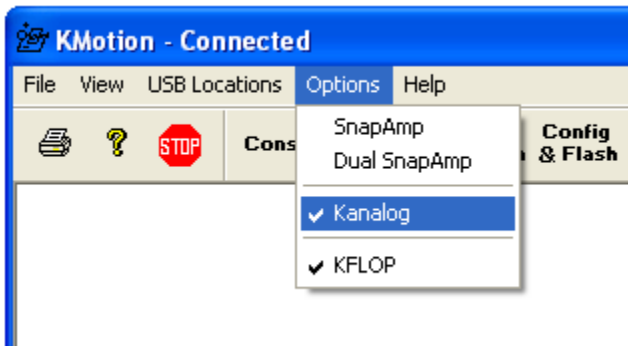


Figure 1 - Kanalog + KFLOP

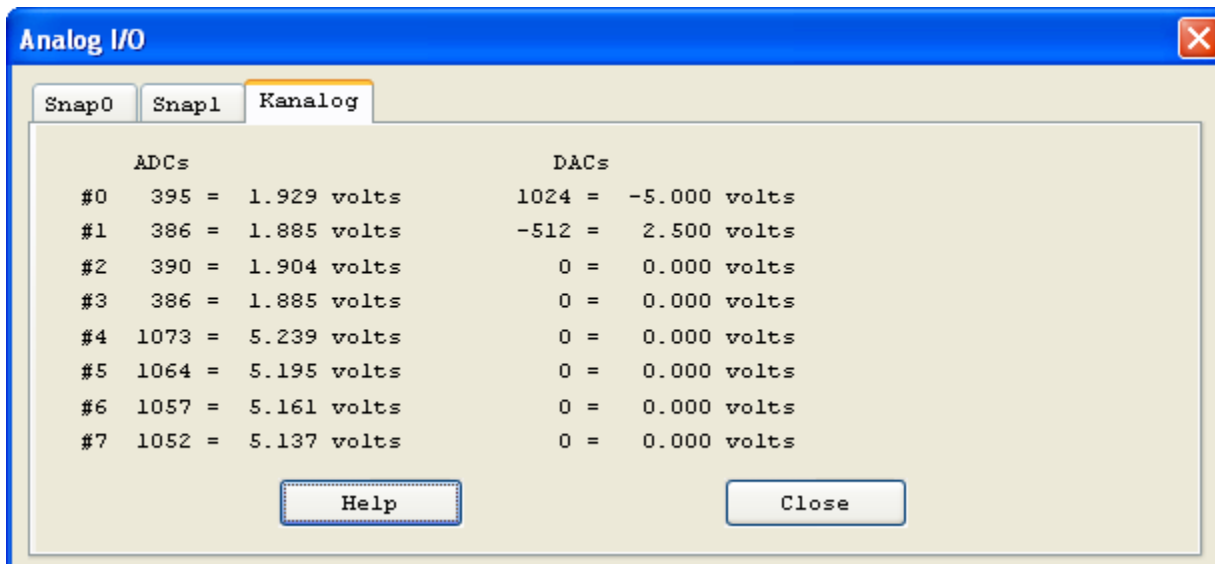
Setting Options



Set options on the KMotion Executive Program for both Kanalog and the required KFlop.

Analog Status

All of the Analog ADC readings and DAC settings can be observed by selecting the Kanalog Tab of the Analog I/O Screen.



Digital Status

All of the Digital I/O can be observed by selecting the Kanalog Tab of the Digital I/O Screen. Inputs are all on the left side of the screen and Outputs are on the right. Outputs may be toggled by clicking on the state.

Bit	State	Bit	State
128 Gen Purpose IN 0	<input checked="" type="checkbox"/>	144 Opto OUT 0	<input type="checkbox"/>
129 Gen Purpose IN 1	<input checked="" type="checkbox"/>	145 Opto OUT 1	<input type="checkbox"/>
130 Gen Purpose IN 2	<input checked="" type="checkbox"/>	146 Opto OUT 2	<input type="checkbox"/>
131 Gen Purpose IN 3	<input checked="" type="checkbox"/>	147 Opto OUT 3	<input type="checkbox"/>
132 Gen Purpose IN 4	<input checked="" type="checkbox"/>	148 Opto OUT 4	<input type="checkbox"/>
133 Gen Purpose IN 5	<input checked="" type="checkbox"/>	149 Opto OUT 5	<input type="checkbox"/>
134 Gen Purpose IN 6	<input checked="" type="checkbox"/>	150 Opto OUT 6	<input type="checkbox"/>
135 Gen Purpose IN 7	<input checked="" type="checkbox"/>	151 Opto OUT 7	<input type="checkbox"/>
136 Opto IN 0	<input type="checkbox"/>	152 FET Driver 0	<input type="checkbox"/>
137 Opto IN 1	<input type="checkbox"/>	153 FET Driver 1	<input type="checkbox"/>
138 Opto IN 2	<input type="checkbox"/>	154 FET Driver 2	<input type="checkbox"/>
139 Opto IN 3	<input type="checkbox"/>	155 FET Driver 3	<input type="checkbox"/>
140 Opto IN 4	<input type="checkbox"/>	156 FET Driver 4	<input type="checkbox"/>
141 Opto IN 5	<input type="checkbox"/>	157 FET Driver 5	<input type="checkbox"/>
142 Opto IN 6	<input type="checkbox"/>	158 FET Driver 6	<input type="checkbox"/>
143 Opto IN 7	<input type="checkbox"/>	159 FET Driver 7	<input type="checkbox"/>
		160 Gen Purpose 0	<input type="checkbox"/>
		161 Gen Purpose 1	<input type="checkbox"/>
		162 Gen Purpose 2	<input type="checkbox"/>
		163 Gen Purpose 3	<input type="checkbox"/>
		164 Gen Purpose 4	<input type="checkbox"/>
		165 Gen Purpose 5	<input type="checkbox"/>
		166 Gen Purpose 6	<input type="checkbox"/>
		167 Gen Purpose 7	<input type="checkbox"/>

Help Close

Example Configuration

The example configuration below shows a typical configuration where an external analog motor amplifier is to be used with differential encoder feedback and optically isolated limit switches. Note the areas on the screen circled in red.

The Axis input type has been selected as "Encoder" with the first Input channel set to 0 (Encoders only use one input channel). A differential encoder should then be connected to Kanalog JP3 A0+ A0- B0+ B0-.

The Axis Output type has been selected as "DAC Servo" with the first Output channel set to 0 (DAC Servos only use one output channel). The Motor's Amplifier should then be connected to Kanalog DAC0 JP11 pin 1 (and ground).

Limit Switch Options have selected I/O bits 136 and 137 which are Kanalog Opto Inputs 0 and 1 on JP15 across pins 1&2 and 3&4.

Configuration

Channel

Microstepper Amplitude

Axis Modes

input

output

Max Following Error

Inv Dist Per Cycle

Lead Compensation

Input Channels

	gain	offset
0	<input type="text" value="0"/>	<input type="text" value="-1"/>
1	<input type="text" value="0"/>	<input type="text" value="1"/>

Output Chan

0	<input type="text" value="0"/>
1	<input type="text" value="0"/>

Limit Switch Options

Negative	Positive
<input checked="" type="checkbox"/> Watch Limit	<input checked="" type="checkbox"/> Watch Limit
<input type="checkbox"/> Stop when low	<input type="checkbox"/> Stop when low
bit no. <input type="text" value="136"/>	bit no. <input type="text" value="137"/>

Action

Flash

User Memory

New Version

Recovery

Help

Close

Save Channel Load Channel Download Channel Upload Channel C Code -> Clipboard

Using SnapAmp 1000

SnapAmp is a very high performance, feature rich, efficient amplifier that expands the capability of the KMotion Motion Control System.

A SnapAmp adds:

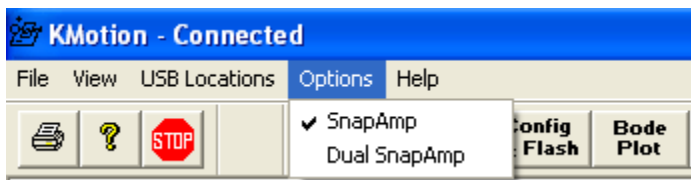
- PWM Amplifier channels
- Opto isolated Inputs
- Differential Encoder inputs
- General Purpose digital IO
- Real-time current measurement for each motor coil
- Real-time power supply voltage measurement
- Programmable peak Supply Current limits
- Programmable power supply voltage clamping
- Digital temperature monitoring

Up to two SnapAmps may be added to a Single KMotion Controller.

A single SnapAmp adds four PWM full bridge amplifiers. The four PWM's are identified as 8,9,10,11 for the first SnapAmp in a system and 12,13,14,15 for a second SnapAmp in a system. A single PWM/Full bridge is required for a brush motor, and a consecutive pair of PWM/Full Bridges are required for a Stepper Motor or Brushless Motor.

A single SnapAmp adds four Quadrature Encoder inputs. The four Quadrature Encoder inputs are identified as 8,9,10,11 for the first SnapAmp in a system and 12,13,14,15 for a second SnapAmp in a system.

Within the KMotion executive program under the option menu set whether there are one or two SnapAmps connected to KMotion. This enables expanded Analog and Digital Screens that will then display the additional I/O available on the SnapAmp(s).



The middle portion of the Analog Status Screen displays the measured currents, supply voltages, temperatures, and current PWM settings.

Status

KMotion

ADCs	DACs	PWMs
#0 -1466 = 7.158 volts	0 = 0.000 volts	0R= 0.0%
#1 -962 = 4.697 volts	0 = 0.000 volts	0R= 0.0%
#2 -930 = 4.541 volts	0 = 0.000 volts	0R= 0.0%
#3 -921 = 4.497 volts	0 = 0.000 volts	0R= 0.0%
#4 -2042 = 0.007 amps	0 = 2.425 volts	0R= 0.0%
#5 -2046 = 0.002 amps	0 = 2.425 volts	0R= 0.0%
#6 -2045 = 0.004 amps	0 = 2.425 volts	0R= 0.0%
#7 -2045 = 0.004 amps	0 = 2.425 volts	0R= 0.0%

Snap 0

ADCs	Supplies	PWMs
#8 8195 = 0.013 amps	#4 35328 = 54.9 V	0 = 0.0%
#9 8555 = 1.577 amps	#5 35190 = 54.7 V	20 = 7.8%
#10 8205 = 0.056 amps	Temperature	0 = 0.0%
#11 8500 = 1.338 amps	#4 202 = 25.3 C	20 = 7.8%
	#5 201 = 25.1 C	

Snap 1

ADCs	Supplies	PWMs
#12 8214 = 0.096 amps	#6 35571 = 55.3 V	0 = 0.0%
#13 8556 = 1.582 amps	#7 35574 = 55.3 V	20 = 7.8%
#14 8192 = 0.000 amps	Temperature	0R= 0.0%
#15 8192 = 0.000 amps	#6 213 = 26.6 C	0R= 0.0%
	#7 183 = 22.9 C	

Dest	Position	Enable	Modes	Done
#0	0.00	#0 0	<input type="checkbox"/> Encoder Microstep	<input checked="" type="checkbox"/> <input type="checkbox"/>
#1	0.00	#1 0	<input type="checkbox"/> Encoder Microstep	<input checked="" type="checkbox"/> <input type="checkbox"/>
#2	0.00	#2 0	<input type="checkbox"/> Encoder Microstep	<input checked="" type="checkbox"/> <input type="checkbox"/>
#3	0.00	#3 0	<input type="checkbox"/> Encoder Microstep	<input checked="" type="checkbox"/> <input type="checkbox"/>
#4	0.00	#4 0	<input checked="" type="checkbox"/> No Input Microstep	<input checked="" type="checkbox"/> <input type="checkbox"/>
#5	0.00	#5 0	<input checked="" type="checkbox"/> No Input Microstep	<input checked="" type="checkbox"/> <input type="checkbox"/>
#6	0.00	#6 0	<input checked="" type="checkbox"/> No Input Microstep	<input checked="" type="checkbox"/> <input type="checkbox"/>
#7	0.00	#7 0	<input type="checkbox"/> Encoder Microstep	<input checked="" type="checkbox"/> <input type="checkbox"/>

Help Close

The Digital I/O screen displays the IO bits (numbers 64 - 93) for SnapAmp 0 in the middle portion of the screen, and IO bits numbered (96 - 125) for Snap Amp 1 on the right portion of the screen. The original KMotion I/O bits remain on the left portion of the screen. Note that KMotion I/O Bits 20-28 are used for the high speed communication to the SnapAmps and may not be used as user IO.

KMotion				SnapAmp 0				SnapAmp 1						
Output	Bit	State	Output Bit	State	Bit	State	Output	Bit	State	Bit	State	Output	Bit	State
<input type="checkbox"/>	0 Enc 0 A	<input type="checkbox"/>	<input checked="" type="checkbox"/>	20	<input checked="" type="checkbox"/>	<input type="checkbox"/>	64 Diff 0 Enc 0 A	<input checked="" type="checkbox"/>	<input type="checkbox"/>	80 GPIO 0	<input type="checkbox"/>	<input type="checkbox"/>	96 Diff 0 Enc 0 A	<input checked="" type="checkbox"/>
<input type="checkbox"/>	1 Enc 0 B	<input type="checkbox"/>	<input checked="" type="checkbox"/>	21	<input type="checkbox"/>	<input type="checkbox"/>	65 Diff 1 Enc 0 B	<input checked="" type="checkbox"/>	<input type="checkbox"/>	81 GPIO 1	<input type="checkbox"/>	<input type="checkbox"/>	97 Diff 1 Enc 0 B	<input checked="" type="checkbox"/>
<input type="checkbox"/>	2 Enc 1 A	<input type="checkbox"/>	<input checked="" type="checkbox"/>	22	<input type="checkbox"/>	<input type="checkbox"/>	66 Diff 2 Enc 1 A	<input checked="" type="checkbox"/>	<input type="checkbox"/>	82 GPIO 2	<input type="checkbox"/>	<input type="checkbox"/>	98 Diff 2 Enc 1 A	<input checked="" type="checkbox"/>
<input type="checkbox"/>	3 Enc 1 B	<input type="checkbox"/>	<input checked="" type="checkbox"/>	23	<input type="checkbox"/>	<input type="checkbox"/>	67 Diff 3 Enc 1 B	<input checked="" type="checkbox"/>	<input type="checkbox"/>	83 GPIO 3	<input type="checkbox"/>	<input type="checkbox"/>	99 Diff 3 Enc 1 B	<input checked="" type="checkbox"/>
<input type="checkbox"/>	4 Enc 2 A	<input type="checkbox"/>	<input checked="" type="checkbox"/>	24	<input type="checkbox"/>	<input type="checkbox"/>	68 Diff 4 Enc 2 A	<input checked="" type="checkbox"/>	<input type="checkbox"/>	84 GPIO 4	<input type="checkbox"/>	<input type="checkbox"/>	100 Diff 4 Enc 2 A	<input checked="" type="checkbox"/>
<input type="checkbox"/>	5 Enc 2 B	<input type="checkbox"/>	<input type="checkbox"/>	25	<input type="checkbox"/>	<input type="checkbox"/>	69 Diff 5 Enc 2 B	<input checked="" type="checkbox"/>	<input type="checkbox"/>	85 GPIO 5	<input type="checkbox"/>	<input type="checkbox"/>	101 Diff 5 Enc 2 B	<input checked="" type="checkbox"/>
<input type="checkbox"/>	6 Enc 3 A	<input type="checkbox"/>	<input type="checkbox"/>	26	<input type="checkbox"/>	<input type="checkbox"/>	70 Diff 6 Enc 3 A	<input checked="" type="checkbox"/>	<input type="checkbox"/>	86 GPIO 6	<input type="checkbox"/>	<input type="checkbox"/>	102 Diff 6 Enc 3 A	<input checked="" type="checkbox"/>
<input type="checkbox"/>	7 Enc 3 B	<input type="checkbox"/>	<input checked="" type="checkbox"/>	27	<input checked="" type="checkbox"/>	<input type="checkbox"/>	71 Diff 7 Enc 3 B	<input checked="" type="checkbox"/>	<input type="checkbox"/>	87 GPIO 7	<input type="checkbox"/>	<input type="checkbox"/>	103 Diff 7 Enc 3 B	<input checked="" type="checkbox"/>
<input type="checkbox"/>	8 Home 0	<input type="checkbox"/>				<input type="checkbox"/>	72 Opto 0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	88 GPIO 8	<input type="checkbox"/>	<input type="checkbox"/>	104 Opto 0	<input checked="" type="checkbox"/>
<input type="checkbox"/>	9 Home 1	<input type="checkbox"/>	28 +/-15V	<input type="checkbox"/>		<input type="checkbox"/>	73 Opto 1	<input checked="" type="checkbox"/>	<input type="checkbox"/>	89 GPIO 9	<input type="checkbox"/>	<input type="checkbox"/>	105 Opto 1	<input checked="" type="checkbox"/>
<input type="checkbox"/>	10 Home 2	<input type="checkbox"/>	29 FAN	<input type="checkbox"/>		<input type="checkbox"/>	74 Opto 2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	90 GPIO 10	<input type="checkbox"/>	<input type="checkbox"/>	106 Opto 2	<input checked="" type="checkbox"/>
<input type="checkbox"/>	11 Home 3	<input type="checkbox"/>	30 AUX SW	<input type="checkbox"/>		<input type="checkbox"/>	75 Opto 3	<input checked="" type="checkbox"/>	<input type="checkbox"/>	91 GPIO 11	<input type="checkbox"/>	<input type="checkbox"/>	107 Opto 3	<input checked="" type="checkbox"/>
<input type="checkbox"/>	12 LIM + 0	<input type="checkbox"/>	31 Thermal	<input type="checkbox"/>		<input type="checkbox"/>	76 Opto 4	<input checked="" type="checkbox"/>	<input type="checkbox"/>	92 GPIO 12	<input type="checkbox"/>	<input type="checkbox"/>	108 Opto 4	<input checked="" type="checkbox"/>
<input type="checkbox"/>	13 LIM - 0	<input type="checkbox"/>	Warning	<input type="checkbox"/>		<input type="checkbox"/>	77 Opto 5	<input checked="" type="checkbox"/>	<input type="checkbox"/>	93 GPIO 13	<input type="checkbox"/>	<input type="checkbox"/>	109 Opto 5	<input checked="" type="checkbox"/>
<input type="checkbox"/>	14 LIM + 1	<input type="checkbox"/>				<input type="checkbox"/>	78 Opto 6	<input checked="" type="checkbox"/>				<input type="checkbox"/>	110 Opto 6	<input checked="" type="checkbox"/>
<input type="checkbox"/>	15 LIM - 1	<input type="checkbox"/>				<input type="checkbox"/>	79 Opto 7	<input checked="" type="checkbox"/>				<input type="checkbox"/>	111 Opto 7	<input checked="" type="checkbox"/>
<input type="checkbox"/>	16 LIM + 2	<input type="checkbox"/>												
<input type="checkbox"/>	17 LIM - 2	<input type="checkbox"/>												
<input type="checkbox"/>	18 LIM + 3	<input type="checkbox"/>												
<input type="checkbox"/>	19 LIM - 3	<input type="checkbox"/>												

SnapAmp has programmable peak current limiting and also programmable peak supply voltage clamping. The clamping is required for large machines because when a massive machine stops quickly, the mechanical energy ends up getting injected back into the power supply. Some power supplies don't like this and allow their voltage to rise up possibly causing damage to the supply or the SnapAmp. The clamping feature protects against this. Currently there isn't a way to configure the current limits and voltage clamp from the KMotion Screens. Listed below is a small C program that may be used to set them appropriately for most systems. Snap Amp has 2 green LEDs. One just blinks to say it is alive and running. The other turns on for a fault condition. A fault can be an over current or over temp. When a fault condition is present, all amps are disabled. When you power up KMotion the fault LED should be on until you run the program and the current limit is set.

SnapAmp peak current limiting is measured from the Motor Supply high side terminal. High side supply measurement is preferred for short protection because most wiring shorts to ground are detected and properly trigger a fault. SnapAmp measures both High side and Low side currents. The Low Side current measurement is made specific to each motor lead, digitized with a precision 10-bit ADC and available for plotting. High Side current measurement is fairly crude with a few programmable thresholds for detecting catastrophic events such as shorts and stalls. Threshold levels 9 through 15 set the approximate fault threshold. The lowest value that allows proper operation of your system without faults should be used. The Table below lists the approximate fault current thresholds for levels 9 through 15. The default power up value of 0 will generate a continuous fault until set to the appropriate value. Note that if *either* of the two power supply peak current detectors are over threshold the entire SnapAmp board (all 4 full bridges) will be disabled. When a fault occurs all full bridges will be disabled for approximately 1/4 second, after which the board will be automatically re-enabled until another fault is detected. *Note: Since any over current fault disables the amplifier for 1/4 second, whenever a fault is cleared, including when the level is initially changed from zero, a delay of 1/2 second should be made before attempting any motion.*

Approximate Peak Current Thresholds:

Level	Amps
9	2
10	6
11	10
12	14
13	18
14	22
15	26

Below is an example C program that must be loaded and executed in the KMotion Board to set the Peak Current limits for each motor supply (to threshold level 9 in this example) and to set the Supply voltage clamp level (to 90V in this example). The Supply Voltage clamping level should be set several volts higher than the actual supply voltage. The example shows setting the values on the first SnapAmp. To set values on the second snap amp replace the symbol SNAP0 with SNAP1.

```
#include "KMotionDef.h"
main()
{
    // peak current limits
    WriteSnapAmp(SNAP0+SNAP_PEAK_CUR_LIMIT0,9);
    WriteSnapAmp(SNAP0+SNAP_PEAK_CUR_LIMIT1,9);
    // clamp supply to 90V
    WriteSnapAmp(SNAP0+SNAP_SUPPLY_CLAMP0
    ,SNAP_CONVERT_VOLTS_TO_ADC(90.0));
    WriteSnapAmp(SNAP0+SNAP_SUPPLY_CLAMP1
    ,SNAP_CONVERT_VOLTS_TO_ADC(90.0));
    // enable supply clamping
    WriteSnapAmp(SNAP0+SNAP_SUPPLY_CLAMP_ENA0 ,1);
    WriteSnapAmp(SNAP0+SNAP_SUPPLY_CLAMP_ENA1 ,1);
}
```

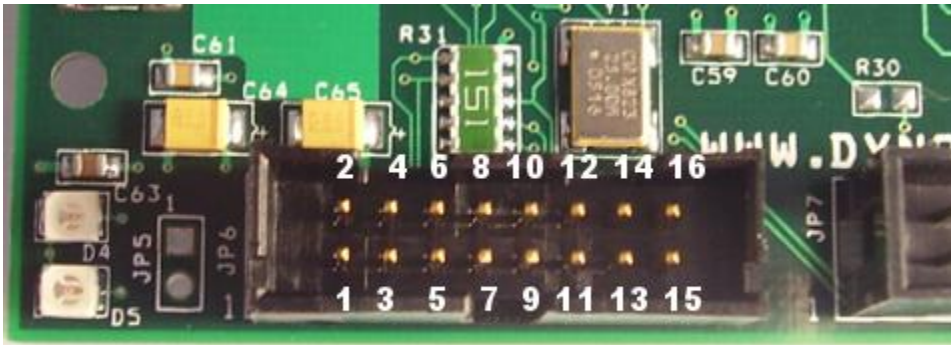
SnapAmp - Connector Pinouts

SnapAmp contains three connectors labeled JP1, JP6, and JP7.

JP6 - KMotion Communication & Logic Power

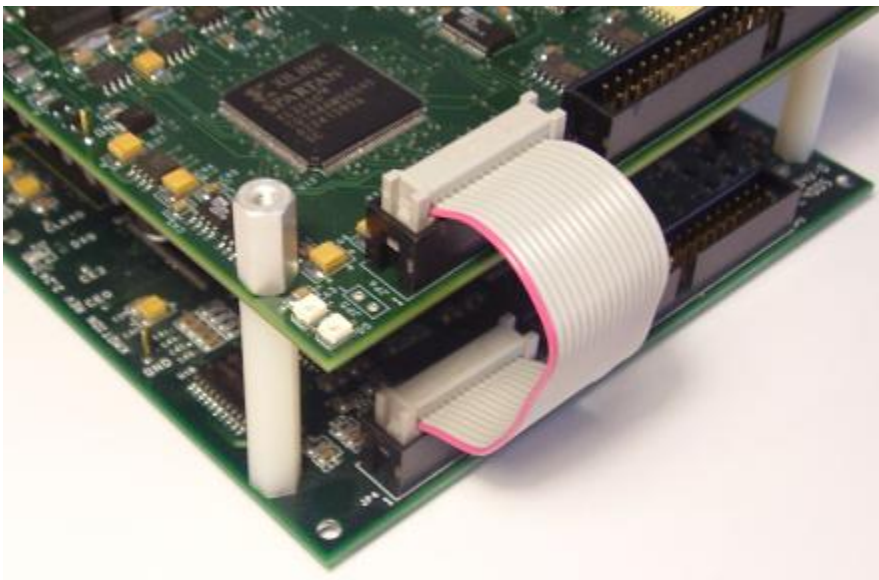
JP6 is a proprietary high-speed communication bus where command and status communication between the KMotion board and the SnapAmp Amplifier. Up to two SnapAmps may be attached to a single KMotion Board. The second of the two SnapAmps must have a configuration jumper installed into JP5.

This connection is required for proper operation of the SnapAmp and should be as short as possible.



16 pin ribbon cable connection between SnapAmp and KMotion.

Note: 4 - 40 x 1 3/8 inch standoffs are used between SnapAmp (top) and KMotion (bottom).



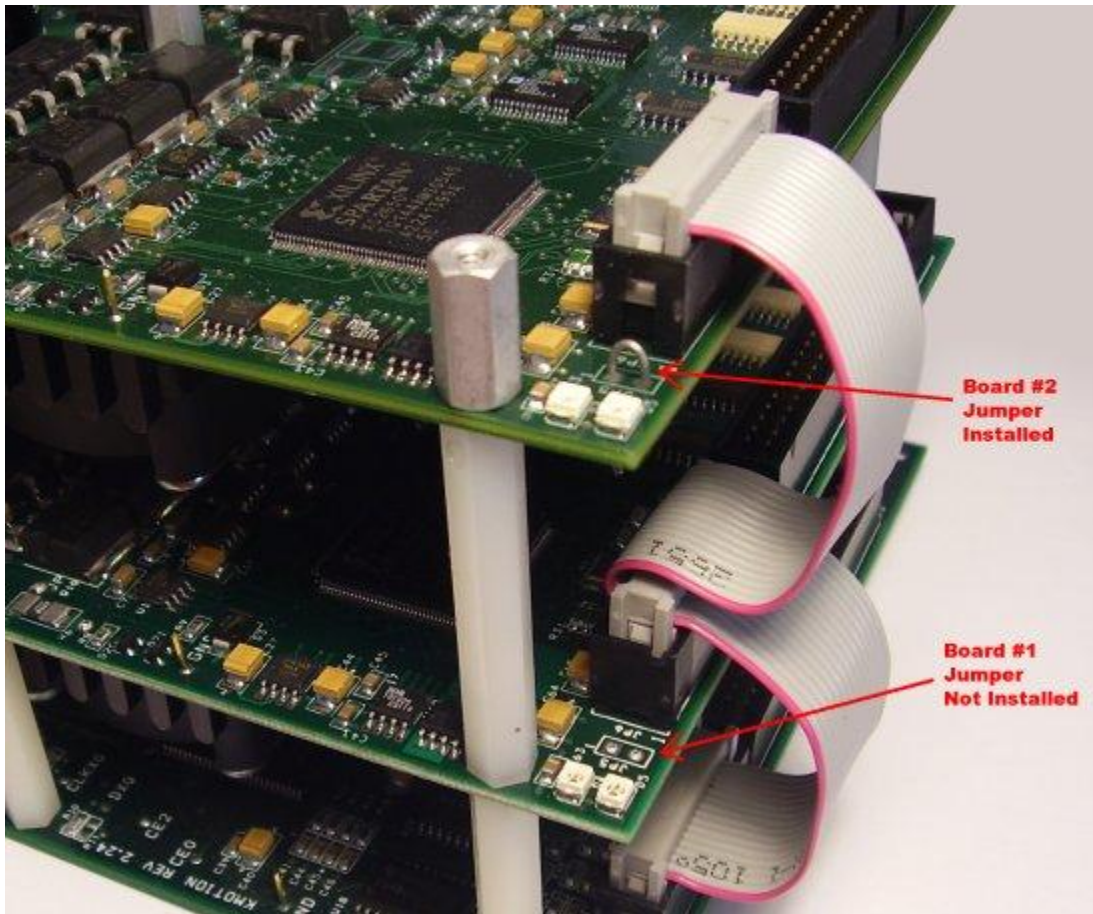
16 pin ribbon cable connection between Dual SnapAmps and KMotion

Notes:

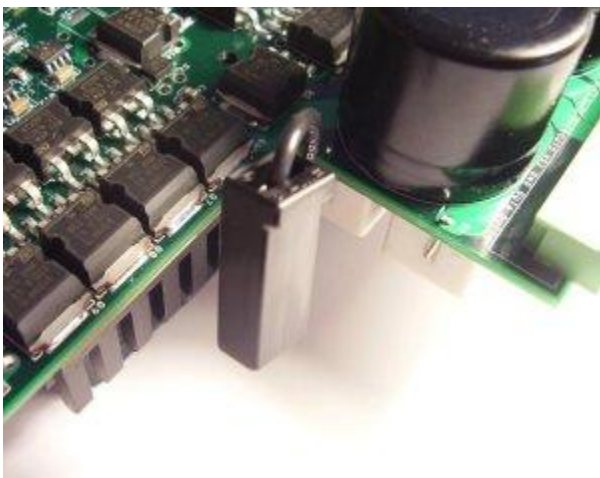
4 - 40 x 1 3/8 inch standoffs are used between SnapAmp and KMotion

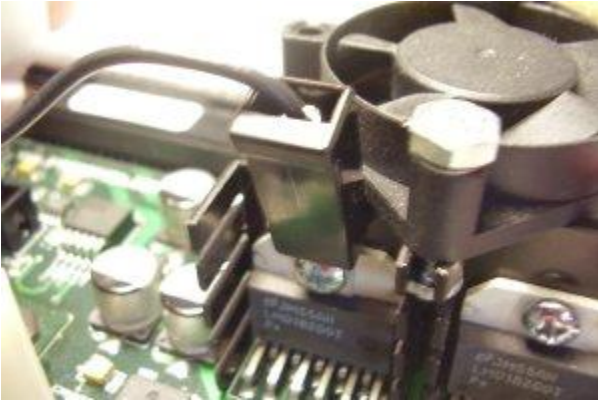
4 - 40 x 1 5/8 inch standoffs are used between SnapAmps

JP5 Jumper installed configures a SnapAmp as the 2nd SnapAmp.

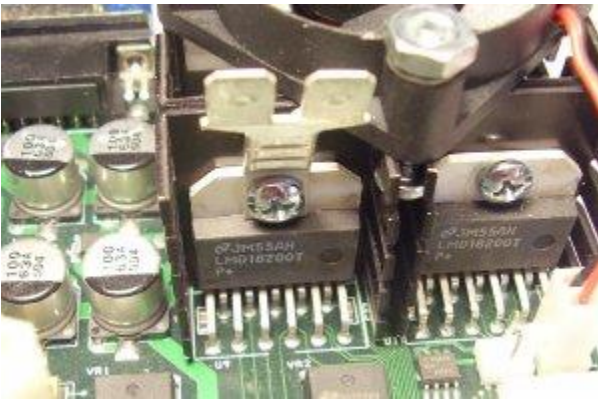


When attaching SnapAmp to the KMotion first attach main ground plug as shown.





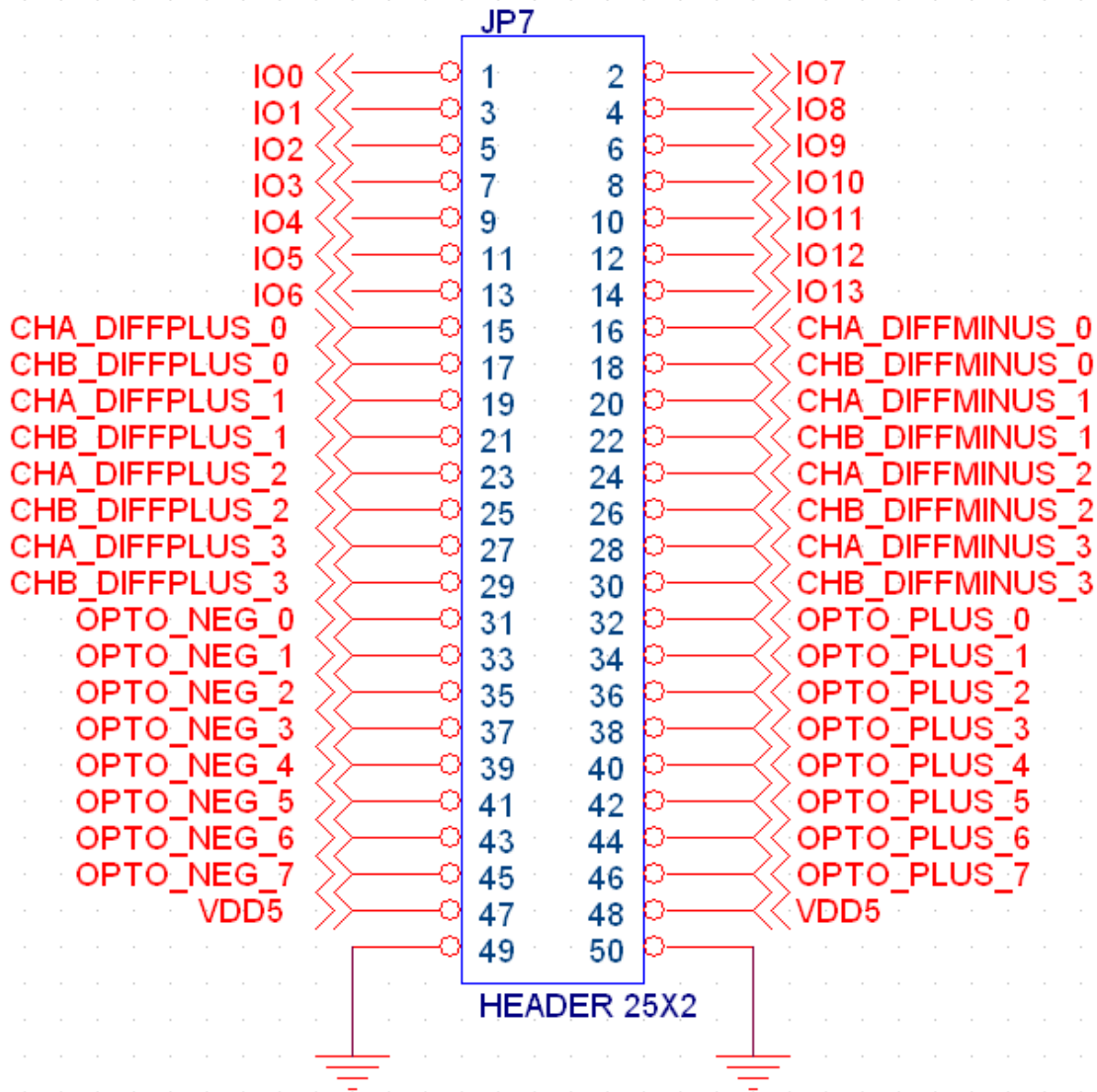
For Dual Snap Amp systems attach the 1:2 spade Y adapter before connecting the first SnapAmp



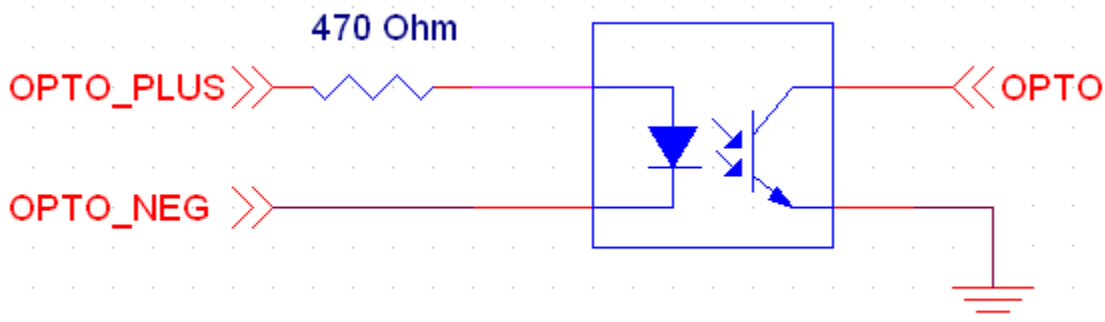
JP7 - I/O - General Purpose LVTTL - OPTO Isolated - Differential - Encoder Inputs

JP7 Is used for all Digital I/O. Fourteen General Purpose LVTTL I/O, Eight Differential Encoder Inputs, and Eight Optically Isolated Inputs.





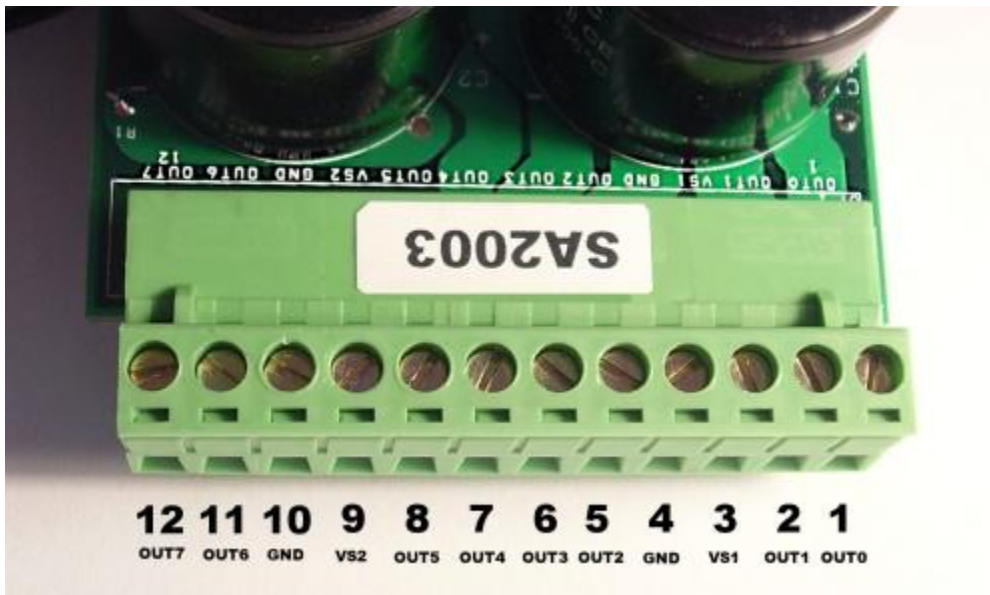
Optically isolated input circuit. 5-12V may be applied. Current requirements at 5V is approximately 6ma and at 12V is approximately 20ma.



Pin	Name	Description	Bit (SnapAmp #0)	Bit (SnapAmp #1)
1	GPIO0	Gen Purpose LVTTL	80	112
2	GPIO1	Gen Purpose LVTTL	81	113
3	GPIO2	Gen Purpose LVTTL	82	114
4	GPIO3	Gen Purpose LVTTL	83	115
5	GPIO4	Gen Purpose LVTTL	84	116
6	GPIO5	Gen Purpose LVTTL	85	117
7	GPIO6	Gen Purpose LVTTL	86	118
8	GPIO7	Gen Purpose LVTTL	87	119
9	GPIO8	Gen Purpose LVTTL	88	120
10	GPIO9	Gen Purpose LVTTL	89	121
11	GPIO10	Gen Purpose LVTTL	90	122
12	GPIO11	Gen Purpose LVTTL	91	123
13	GPIO12	Gen Purpose LVTTL	92	124
14	GPIO13	Gen Purpose LVTTL	93	125
15	CHA DIFF PLUS 0	Differential Input + Encoder 0 Input Phase A	64	96
16	CHA DIFF MINUS 0	Differential Input - Encoder 0 Input Phase A	64	96
17	CHB DIFF PLUS 0	Differential Input + Encoder 0 Input Phase B	65	97
18	CHB DIFF MINUS 0	Differential Input - Encoder 0 Input Phase B	65	97
19	CHA DIFF PLUS 1	Differential Input + Encoder 1 Input Phase A	66	98
20	CHA DIFF MINUS 1	Differential Input - Encoder 1 Input Phase A	66	98
21	CHB DIFF PLUS 1	Differential Input + Encoder 1 Input Phase B	67	99
22	CHB DIFF MINUS 1	Differential Input - Encoder 1 Input Phase B	67	99
23	CHA DIFF PLUS 2	Differential Input + Encoder 2 Input Phase A	68	100
24	CHA DIFF MINUS 2	Differential Input - Encoder 2 Input Phase A	68	100
25	CHB DIFF PLUS 2	Differential Input + Encoder 2 Input Phase B	69	101
26	CHB DIFF MINUS 2	Differential Input - Encoder 2 Input Phase B	69	101
27	CHA DIFF PLUS 3	Differential Input + Encoder 3 Input Phase A	70	102
28	CHA DIFF MINUS 3	Differential Input - Encoder 3 Input Phase A	70	102
29	CHB DIFF PLUS 3	Differential Input + Encoder 3 Input Phase B	71	103

	3	Phase B		
30	CHB DIFF MINUS 3	Differential Input - Encoder 3 Input Phase B	71	103
31	OPTO NEG 0	Opto Isolated Input 0 Negative Connection	72	104
32	OPTO POS 0	Opto Isolated Input 0 Positive Connection	72	104
33	OPTO NEG 1	Opto Isolated Input 1 Negative Connection	73	105
34	OPTO POS 1	Opto Isolated Input 1 Positive Connection	73	105
35	OPTO NEG 2	Opto Isolated Input 2 Negative Connection	74	106
36	OPTO POS 2	Opto Isolated Input 2 Positive Connection	74	106
37	OPTO NEG 3	Opto Isolated Input 3 Negative Connection	75	107
38	OPTO POS 3	Opto Isolated Input 3 Positive Connection	75	107
39	OPTO NEG 4	Opto Isolated Input 4 Negative Connection	76	108
40	OPTO POS 4	Opto Isolated Input 4 Positive Connection	76	108
41	OPTO NEG 5	Opto Isolated Input 5 Negative Connection	77	109
42	OPTO POS 5	Opto Isolated Input 5 Positive Connection	77	109
43	OPTO NEG 6	Opto Isolated Input 6 Negative Connection	78	110
44	OPTO POS 6	Opto Isolated Input 6 Positive Connection	78	110
45	OPTO NEG 7	Opto Isolated Input 7 Negative Connection	79	111
46	OPTO POS 7	Opto Isolated Input 7 Positive Connection	79	111
47	VDD5	+ 5V Encoder Power Output		
48	VDD5	+ 5V Encoder Power Output		
49	GND	Digital Logic Ground		
50	GND	Digital Logic Ground		

For a single page printable PDF of the above click [here](#)

JP1 - Motor/Motor Supply (10-80V) Connector

	Motor type DC - Brush	Motor type - 3 Phase brushless	Motor type - Stepper
Axis 0	Connect Motor across OUT0-OUT1 Specify PWM Channel 8	Connect Phase A to OUT0 Connect Phase B to OUT1 Connect Phase C to OUT2 Leave OUT3 disconnected Specify PWM Channel 8	Connect Coil A across OUT0-OUT1 Connect Coil B across OUT2-OUT3 Specify PWM Channels 8 and 9
Axis 1	Connect Motor across OUT2-OUT3 Specify PWM Channel 9		
Axis 2	Connect Motor across OUT4-OUT5 Specify PWM Channel 10	Connect Phase A to OUT4 Connect Phase B to OUT5 Connect Phase C to OUT6 Leave OUT7 disconnected Specify PWM Channel 10	Connect Coil A across OUT4-OUT5 Connect Coil B across OUT6-OUT7 Specify PWM Channels 10 and 11
Axis 3	Connect Motor across OUT6-OUT7 Specify PWM Channel 11		

Configuring Step and Direction Outputs

KFLOP supports connection to motor amplifiers that utilize step and direction (or [quadrature](#)) inputs to control the motion. Step and direction motor amplifiers are typically used with stepper motors with or without micro-stepping capability. Each "step" causes the motor amplifier to advance the motor position one step in the direction specified by the "Direction" signal. Only two digital output signals are required for this interface. Fine micro-stepping resolution combined with high motor RPM can result in step rates in the megahertz. For example, a 200 step/rev stepping motor with 128:1 micro-stepping running at 3000 RPM requires:

$$3000 \text{ RPM} / 60 \text{ Sec/Min} * 200 \text{ steps/rev} * 128 \text{ } \mu\text{steps/full step} = 1.28 \text{ MHz } \mu\text{steps/sec}$$

KFLOP has 8 axes of Step and Direction. Output pulses up to 2.5 MHz can be generated.

The 8 Axes of Step/Dir outputs are normally hard wired to IO bits 8 through 15 on JP7 and IO bits 36 through 43 on JP5. However the first 4 Step/Dir outputs can be multiplexed to connectors JP4 and JP6 if desired. This may be required if JP7 is used for some other purpose such as interfacing to the Kanalog I/O Expander. A global [multiplexing bit](#) is used to switch the outputs to the alternate connectors.

Note that the first 8 of 10 I/O pins of Aux #0 and Aux #1 have internal 150 Ohm pull down resistors. Therefore Pins JP4-13, JP4-14, JP6-13 and JP6-14 may not be used in Open Collector mode.

If an axis channel is selected as a Step and Direction axis, the corresponding 2 output pins will be automatically configured as outputs and they may not be used as general purpose IO.

For KFLOP see the Table below:

Signal	Mux = 0		Mux = 1	
	IO Bit	Pin	IO Bit	Pin
Step 0	8	JP7 - 15	22	JP4 - 13
Direction 0	9	JP7 - 16	23	JP4 - 14
Step 1	10	JP7 - 17	24	JP4 - 15
Direction 1	11	JP7 - 18	25	JP4 - 16
Step 2	12	JP7 - 19	32	JP6 - 13
Direction 2	13	JP7 - 20	33	JP6 - 14
Step 3	14	JP7 - 21	34	JP6 - 15
Direction 3	15	JP7 - 22	35	JP6 - 16

Signal	IO Bit	JP5 Pin
Step 4	36	JP5 - 1
Direction 4	37	JP5 - 2
Step 5	38	JP5 - 3
Direction 5	39	JP5 - 4
Step 6	40	JP5 - 5
Direction 6	41	JP5 - 6
Step 7	42	JP5 - 7
Direction 7	43	JP5 - 8

To configure an axis as step and direction, on the Configuration Screen select output mode as "Step Dir" and select which of the 4 available Step and Direction generators should be used by setting the appropriate Output Channel 0 setting. See below.

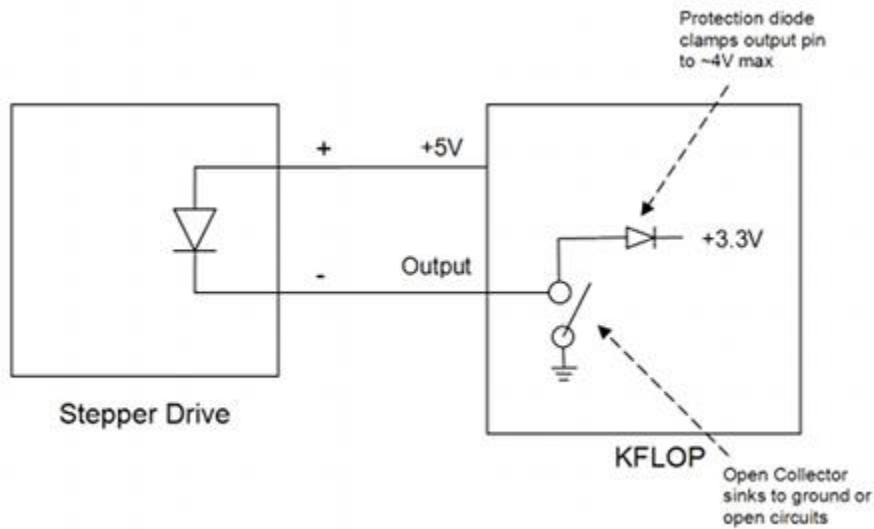
The screenshot shows the 'Configuration' window for KFLOP. The 'Channel' is set to 0. Under 'Axis Modes', the 'output' dropdown is set to 'Step Dir' (highlighted with a red circle). The 'Output Chan' dropdown for channel 0 is also set to 0 (highlighted with a red circle). The 'Input Channels' section shows gain and offset settings for channels 0 and 1. The 'Limit Switch Options' section includes checkboxes for 'Watch Limit' and 'Stop when low' for both negative and positive limits, with 'bit no.' set to 0. The 'Action' dropdown is set to 'Kill Motor Drive'. On the right, there are checkboxes for 'Launch on Power Up' for threads 1 through 7, and a 'Flash' section with buttons for 'User Memory', 'New Version', and 'Recovery'. At the bottom, there are buttons for 'Save Channel', 'Load Channel', 'Download Channel' (highlighted with a dashed border), 'Upload Channel', and 'C Code -> Clipboard'. A 'Help' button and a 'Close' button are also present.

Valid output channel settings 0-31 are allowed for KFLOP. Although only 8 Step and Direction generators are available for KFLOP by adding 8 and/or 16 to the channel number the mode of the Step/Dir Generator can be changed.

Adding 8 to the channel number uses the same generator to be used except the output pins are actively driven (high and low) as 3.3V LVTTTL signals instead of only driven low as open collector outputs.

Adding 16 to the channel number switches the generator from Step/Dir to Quadrature output mode.

If your amplifier has opto coupler inputs driven off +5V then open-collector mode is likely to work better. The diagram below shows how the open collector mode works driving the LED of an Opto Coupler with the anode connected to +5V.



However if the amplifier has standard logic inputs then LVTTL outputs should work better.

Quadrature outputs are required for some amplifiers. Quadrature outputs output two A B phases instead of Step/Dir signals. An advantage is that only one signal edge transition takes place for each "step" as opposed to a complete pulse.

The table below summarizes the modes:

Output Chan 0 Setting	Step & Dir Generator Selected	Output Drive Type
0	0	Open Collector - Step/Dir
1	1	Open Collector - Step/Dir
2	2	Open Collector - Step/Dir
3	3	Open Collector - Step/Dir
4	4	Open Collector - Step/Dir
5	5	Open Collector - Step/Dir
6	6	Open Collector - Step/Dir
7	7	Open Collector - Step/Dir
8	0	LVTTL - Step/Dir
9	1	LVTTL - Step/Dir
10	2	LVTTL - Step/Dir
11	3	LVTTL - Step/Dir
12	4	LVTTL - Step/Dir
13	5	LVTTL - Step/Dir
14	6	LVTTL - Step/Dir
15	7	LVTTL - Step/Dir

16	0	Open Collector - Quadrature
17	1	Open Collector - Quadrature
18	2	Open Collector - Quadrature
19	3	Open Collector - Quadrature
20	4	Open Collector - Quadrature
21	5	Open Collector - Quadrature
22	6	Open Collector - Quadrature
23	7	Open Collector - Quadrature
24	0	LVTTL - Quadrature
25	1	LVTTL - Quadrature
26	2	LVTTL - Quadrature
27	3	LVTTL - Quadrature
28	4	LVTTL - Quadrature
29	5	LVTTL - Quadrature
30	6	LVTTL - Quadrature
31	7	LVTTL - Quadrature

Note when configuring the motion profile for a Step and Direction Axis the appropriate maximum Velocity, Acceleration, and Jerk should be set in units of micro-steps/sec, micro-steps/sec², and micro-steps/sec³ respectively.

Global Register sets Pulse Width, Polarity, Multiplexor

To change the Step/Dir Pulse width, Step Pulse Polarity, and connector multiplexor for channels 0-4 a programmable register in KFLOP's FPGA may be used.

KFLOP has the capability to program the Step pulse width as a 6-bit value. The default setting is 2us. The pulse length may be adjusted from 1 to 63 of 16.67 MHz clocks. Which corresponds to 60ns to 3.78us. Using a long pulse length limits the maximum frequency that can be generated. For example with the default pulse length of 2us the frequency should not exceed $1/(2 \times 2\mu s) = 250\text{KHz}$.

Setting Bit-6 high of the register can be set high to multiplex Step/Dir generators 0-3 from JP7 to JP4 and JP6.

Setting Bit-7 high will invert the Step Output pulse so that it pulses High rather than Low. Some Amplifiers (Geckos) prefer this mode. If the drive "steps" on the falling edge of the pulse, then this option will provide more setup time for the Direction Signal. A User C Program must be used to change the FPGA register. The following statement should be used:

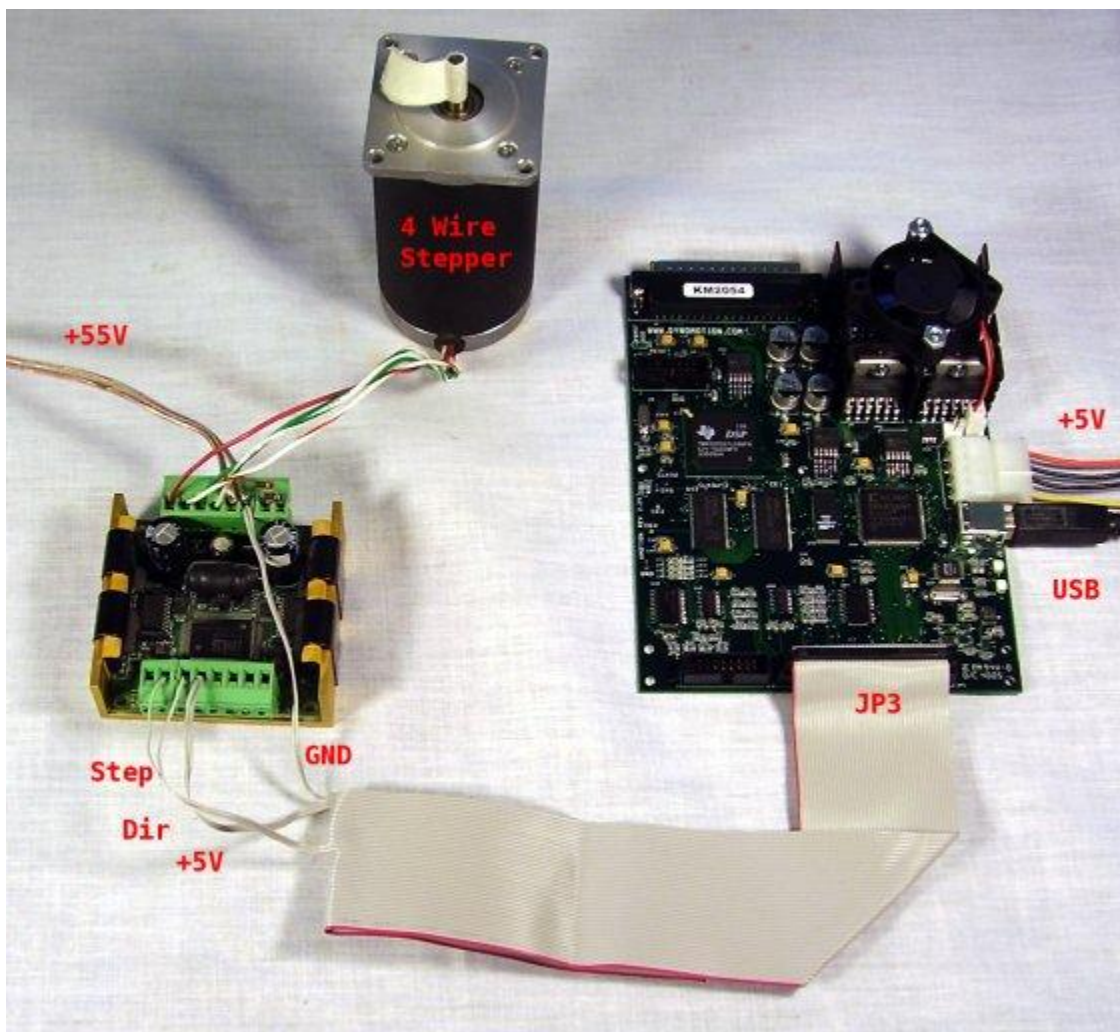
```
FPGA(STEP_PULSE_LENGTH_ADD)=32; // set the pulse time to ~ 2us
```

FPGA(STEP_PULSE_LENGTH_ADD)=32 + 0x40; // set the pulse time to ~ 2us and multiplex to JP4 and JP6

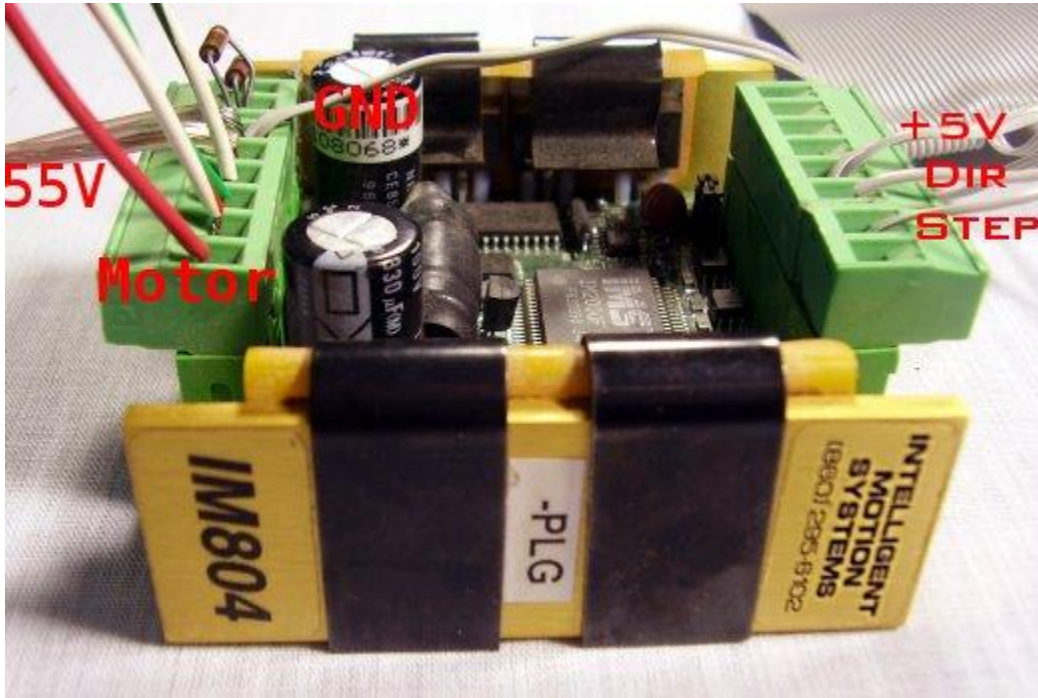
FPGA(STEP_PULSE_LENGTH_ADD)=32 + 0x80; // set the pulse time to ~ 2us and pulse the Step High

FPGA(STEP_PULSE_LENGTH_ADD)=32 + 0x40 + 0x80 // set the pulse time to ~ 2us, mux to JP4 and JP6, and pulse the Step High

Example Configuration for a IM804 Amplifier and Rapidsyn Stepper



Layout - KMotion - Amplifier - Motor



Amplifier Connections - resistors set coil currents (1500 ohms = 3A) and idle standby current (270 ohms = 0.5A). DIP Switches set to ON-OFF-OFF-ON for 128:1 microstepping.



Rapidsyn stepper motor. Note coil winding's centertaps are not used.

Configuration

Channel

Microstepper Amplitude

Max Following Error

Inv Dist Per Cycle

Lead Compensation

Axis Modes

input

output

Input Channels

	gain	offset
0	<input type="text" value="1"/>	<input type="text" value="0"/>
1	<input type="text" value="1"/>	<input type="text" value="0"/>

Output Chan

0	<input type="text" value="0"/>
1	<input type="text" value="1"/>

Limit Switch Options

Negative

☐ Watch Limit

☐ Stop when low

bit no.

Positive

☐ Watch Limit

☐ Stop when low

bit no.

Action

Launch on Power Up

☐ Thread 1

☐ Thread 2

☐ Thread 3

☐ Thread 4

☐ Thread 5

☐ Thread 6

☐ Thread 7

Flash

User Memory

New Version

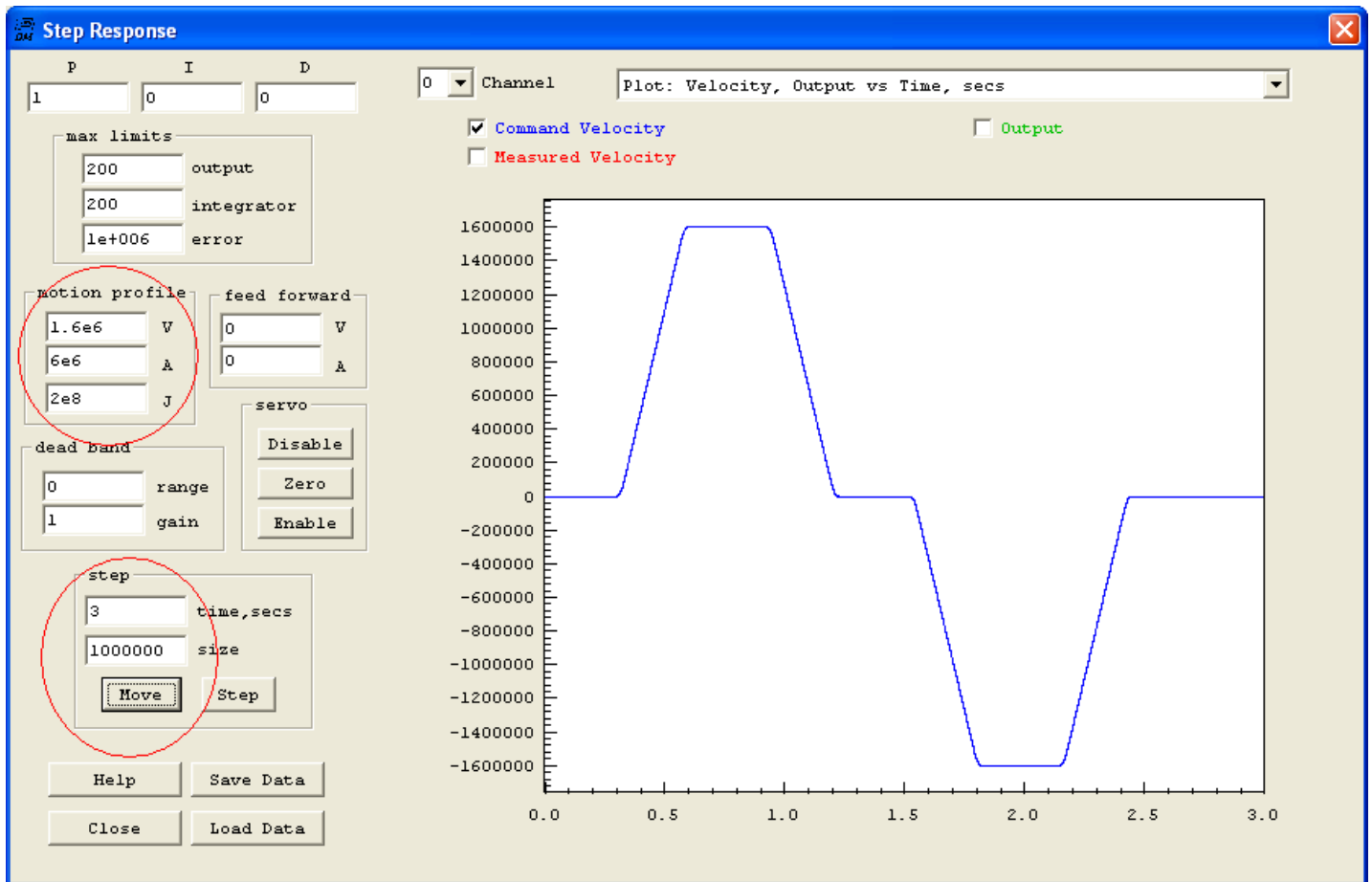
Recovery

Help

Close

Save Channel Load Channel Download Channel Upload Channel C Code -> Clipboard

Axis channel 0 configures as "Step Dir" mode and using Step and Direction generator 0.



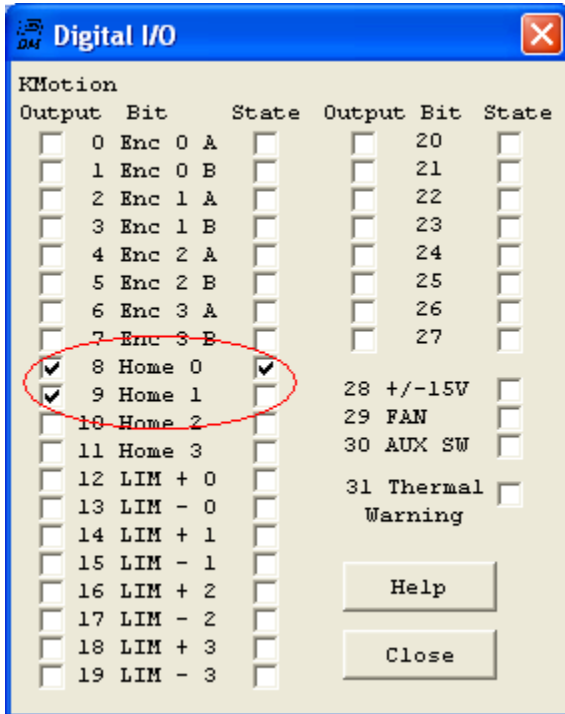
Max Velocity 1.6×10^6 steps/sec ($1.6e6 / 128 = 12,500$ sps = 3750 RPM). Note this (and most) steppers lose most all of their useful torque above several thousand full steps per second. The high speed demonstrated here is intended to demonstrate KMotion's ability to generate high pulse rates.

Max Acceleration 6×10^6 steps/sec²

Max Jerk 2×10^8 steps/sec³

1×10^6 count Move performed and data captured over 3 seconds.

Pushing Move downloads all parameters, enables the axis, performs the movement, and plots the result.



Note after the axis has been enabled, digital IO bits 8 and 9 have automatically been configured as outputs for use by Step and Direction Generator #0.

See Video

Click [here](#) for a video showing slow and smooth acceleration of this configuration up to 12,500 full steps per second (1.6 MHz @ 128 microsteps/full step).

Configuring DC Brush Motor with SnapAmp and Single-Ended Encoder to KMotion

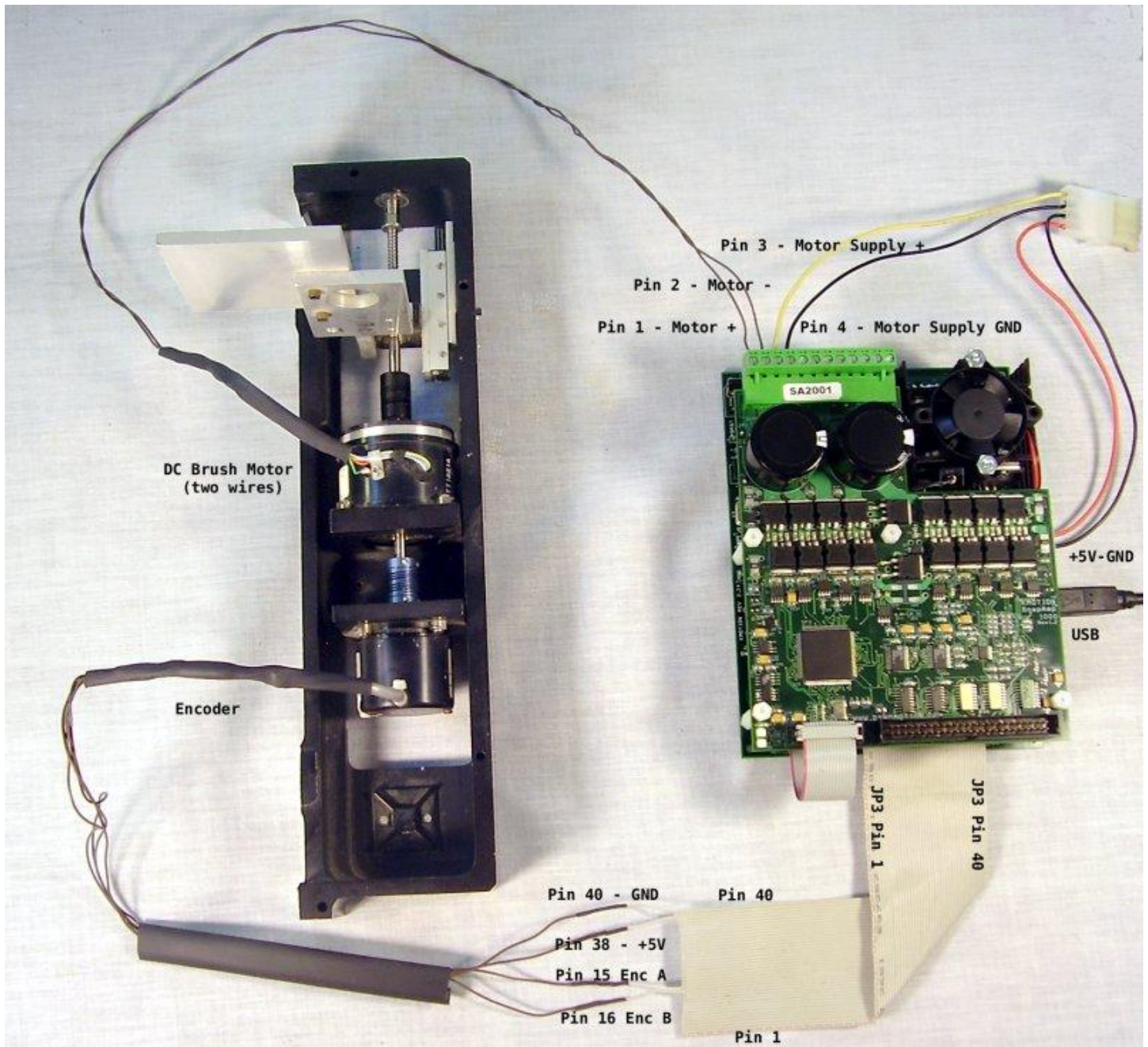
This example will proceed through the following steps

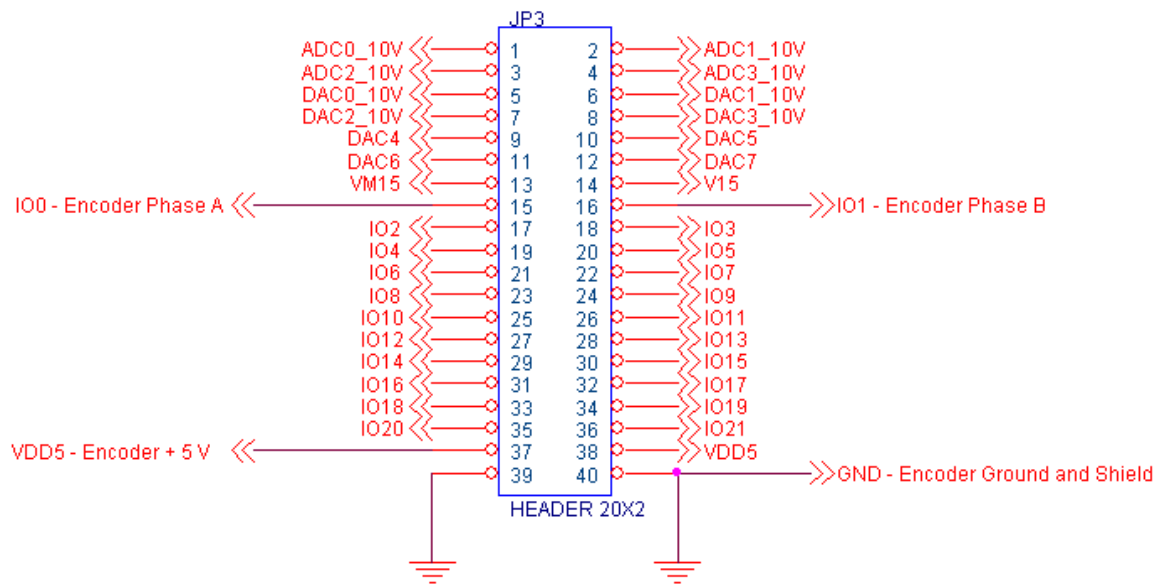
1. [Wiring the Motor, encoder, and Power](#)
2. [Configuring the Software](#)
3. [Testing the encoder](#)
4. [Testing the Motor/Amplifier](#)
5. [Closing the loop](#)
6. [Tuning the Servo](#)
7. [Executing a simple C Program Motion Sequence](#)

[Here is a video overview.](#)

Wiring the Motor, encoder, and Power

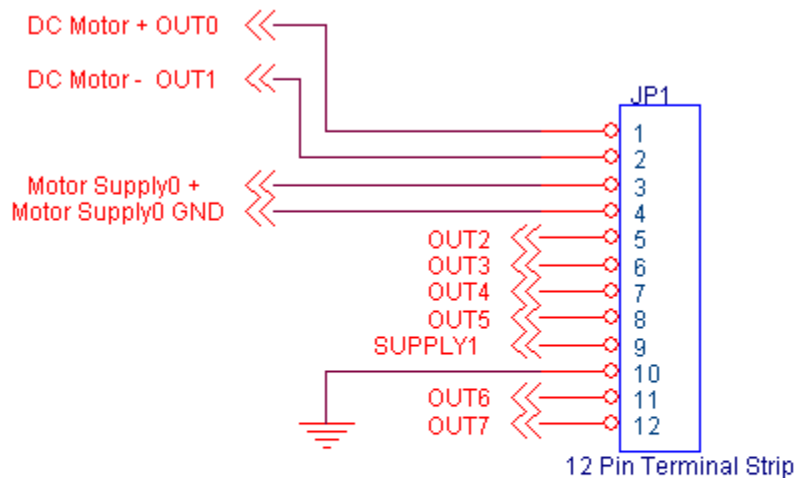
This example configuration shows a single dc brush motor connected to the first of 4 available full bridge drivers on a single SnapAmp 1000. A single-ended encoder is connected to KMotion's JP3 connector as encoder input #0 for Servo feedback. (The KMotion board is hidden underneath the SnapAmp in the photo). Note: Encoders typically come as one of two types - differential outputs (having A+, A-, B+, B-) or single ended outputs (A B). A differential encoder may be used as a single ended encoder by only using the A+ and B+ signals (and leaving the A- and B- unconnected). However single ended signals are more susceptible to noise than differential signals. KMotion has 4 single ended encoder inputs. SnapAmp 1000 has 4 differential encoder inputs. In this example a KMotion single-ended encoder input is used. Some encoders have a index pulse that occurs once per revolution usually labeled as channel Z. In this example the index is not used and is not connected.





Encoder Channel 0 Wiring Diagram

KMotion to Single Ended Encoder



SnapAmp Wiring Diagram

PWM Channel 8 to DC Brush Motor

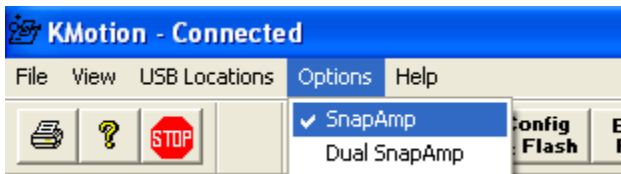
Configuring the Software

After the motor and encoder connections have been made, the software is configured and the encoder feedback and motor is tested.

Although any of KMotion's 8 axes could be used, axis 4 will be configured since axes 0-3 are normally used for KMotion's on-board amplifiers.

In the KMotion Executive program three screens involve setting an axis's configuration: The Configuration Screen, the Step Response Screen, and the Filters Screen. These three screens and our choice of initial settings are shown below:

Execute the KMotion executive program and set the Options for a single SnapAmp. See below:



On the Configuration screen Axis Channel 4 is selected, the input type is set to Encoder at encoder channel 0, the output type is set to DC Servo at PWM channel 8, and a very large following error is set.

Configuration

Channel **4** Microstepper Amplitude **60**

Axis Modes
input **Encoder**
output **DC Servo**

Max Following Error **10000000**

Inv Dist Per Cycle **1**

Lead Compensation **0**

Input Channels

	gain	offset
0 0	1	0
1 4	1	0

Output Chan

0 8	
1 9	

Limit Switch Options

Negative

☐ Watch Limit

☐ Stop when low

bit no. **0**

Positive

☐ Watch Limit

☐ Stop when low

bit no. **0**

Action **Kill Motor Drive**

Launch on Power Up

☐ Thread 1

☐ Thread 2

☐ Thread 3

☐ Thread 4

☐ Thread 5

☐ Thread 6

☐ Thread 7

Flash

User Memory

New Version

Recovery

Help

Close

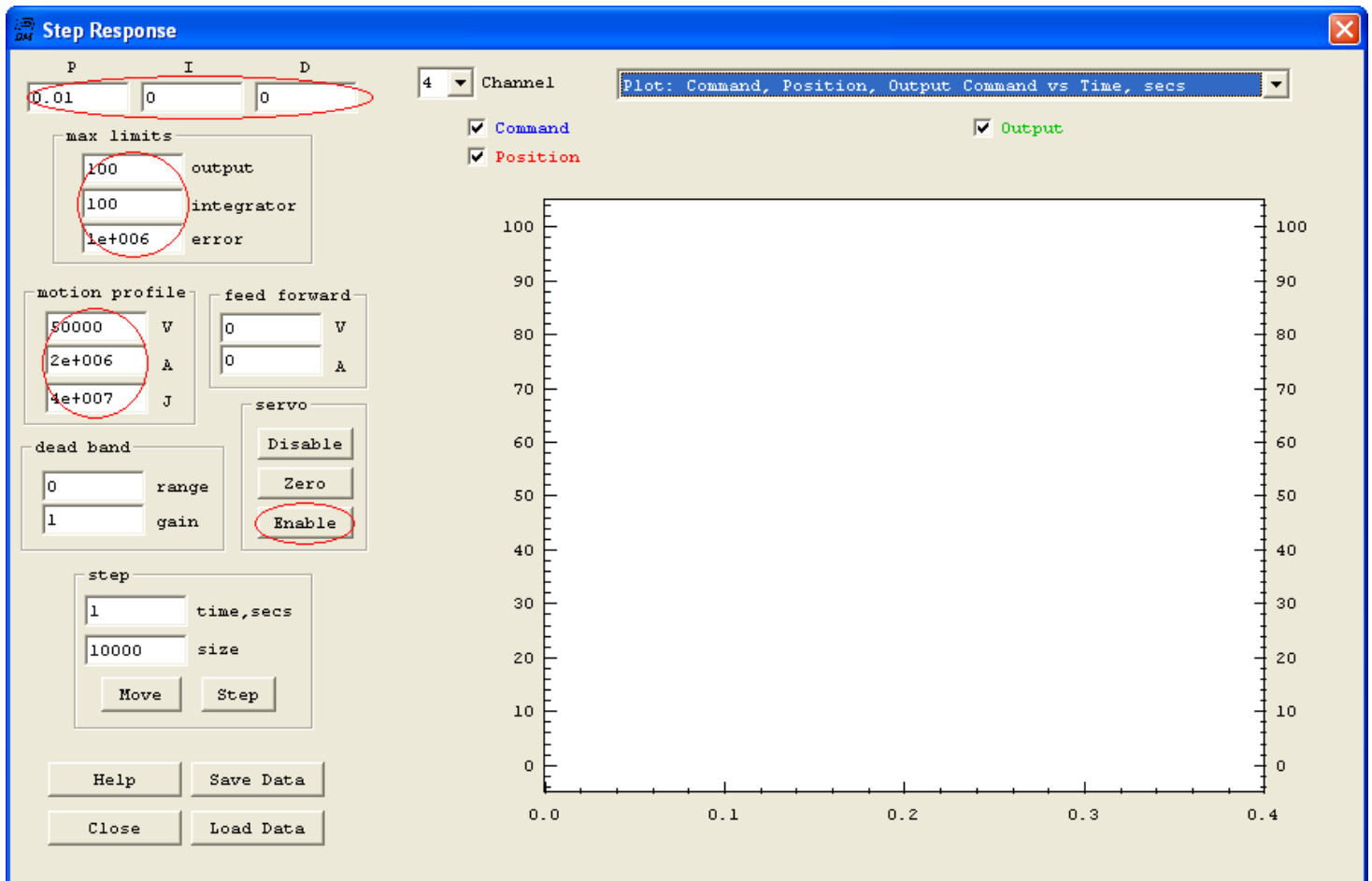
Save Channel Load Channel Download Channel Upload Channel C Code -> Clipboard

On the Filters Screen all filters are disabled by "Clearing" all filters to have a gain of 1.

The screenshot shows the KFLOP Filters screen with three filter panels (Filter 0, Filter 1, Filter 2). Each panel has a 'Z Domain' dropdown menu, a 'Compute' button, and a display for IIR Coefficients. The IIR Coefficients are shown as a transfer function: $y(z) = \frac{B_0 + B_1z^{-1} + B_2z^{-2}}{1 - A_1z^{-1} - A_2z^{-2}}$. Below this, there are input fields for B0, B1, B2, A1, and A2. The 'Clear' button is circled in red in each panel. At the bottom, there is a 'Channel' dropdown set to 4, a 'C Code -> Clipboard' button, a 'Download' button, and 'Help' and 'Close' buttons.

The Step Screen servo loop (PID) parameters are set to only very low proportional gain. The max output is set to a small value that should still allow some motor motion. SnapAmp DC Servo mode outputs current commands in the range of -1000 to +1000 which corresponds to approximately -30Amps to +30 Amps. So a max output of 100 will limit the current to 3 Amps. Although the Integrator will not be initially used we set the max integrator to also be 100. The max error is set to a large number. The motion profile parameters are set to values that we will expect to eventually achieve. Our example uses a low speed torque motor being driven off of only 12V. This encoder has 4000 counts/rev so a speed of 50,000 cnts/sec corresponds to 12.5 rev/sec (750 RPM).

With the Motor Power Supply tuned off, pushing the Enable button will download the parameters from all screens and enable the axis without any motor motion. This will allow us to ensure the axis is properly wired and configured as encoder input and of the correct encoder channel.



Testing the encoder

We can now rotate the motor/encoder shaft by hand and should observe counts on Axis Channel 4 on the Analog Status Screen. Note if one of the other axis's configurations also is defined as input Encoder channel 0 then that axis may count as well. The default configuration for axis 0 is Encoder input from channel 0. To avoid confusion, disable any other axis by selecting that axis, define the input/output types as "No Input" and "No Output", then download, then disable the axis. To test the encoder, rotate the encoder and check if the expected number of counts per rev are obtained. Rotating the axis the opposite direction should count in the opposite direction.

Status

KMotion

ADCs	DACs	PWMs
#0 -1471 = 7.183 volts	0 = 0.000 volts	OR=
#1 -1062 = 5.186 volts	0 = 0.000 volts	OR=
#2 -1035 = 5.054 volts	0 = 0.000 volts	OR=
#3 -1027 = 5.015 volts	0 = 0.000 volts	OR=
#4 -2043 = 0.006 amps	0 = 2.425 volts	OR=
#5 -2046 = 0.002 amps	0 = 2.425 volts	OR=
#6 -2046 = 0.002 amps	0 = 2.425 volts	OR=
#7 -2046 = 0.002 amps	0 = 2.425 volts	OR=

Snap 0

ADCs	Supplies	PWMs
#8 8155 = -0.158 amps	#4 7680 = 11.6 V	OC= 0.0%
#9 8157 = -0.150 amps	#5 825 = 0.9 V	OR= 0.0%
#10 8182 = -0.043 amps	Temperature	OR= 0.0%
#11 8190 = -0.009 amps	#4 277 = 34.6 C	OR= 0.0%
	#5 267 = 33.4 C	

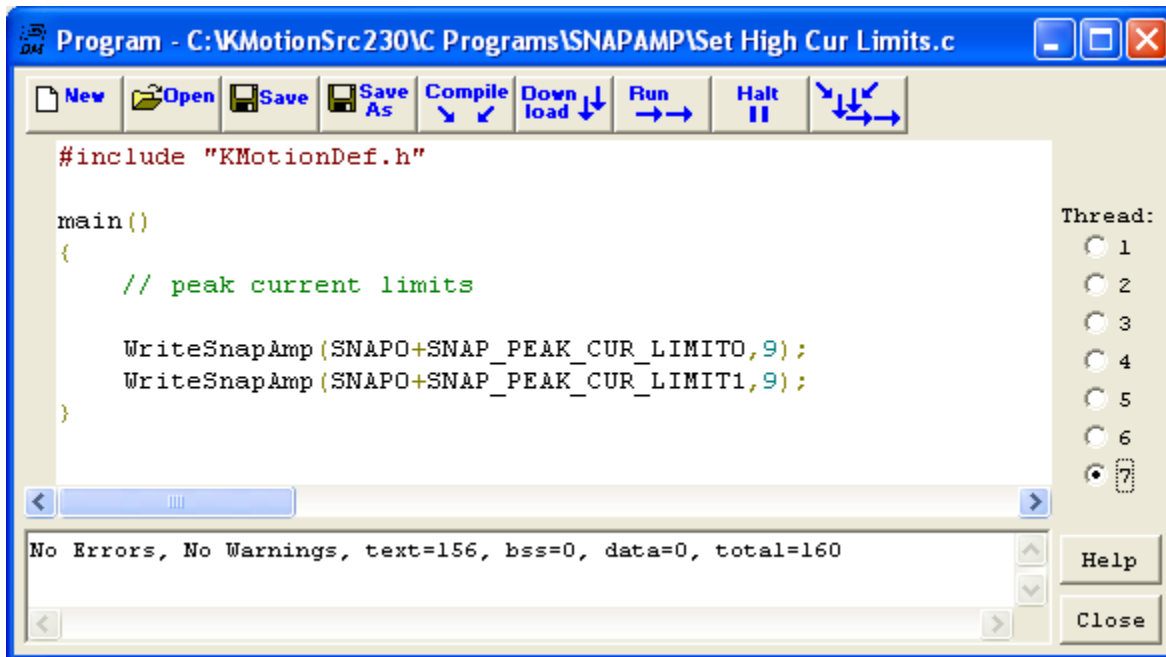
Dest	Position	Enable	Modes	Done
#0	0.00	#0 0	<input type="checkbox"/> No Input <input type="checkbox"/> No Output	<input checked="" type="checkbox"/> <input type="checkbox"/>
#1	0.00	#1 2	<input type="checkbox"/> Encoder <input type="checkbox"/> Microstep	<input checked="" type="checkbox"/> <input type="checkbox"/>
#2	0.00	#2 2	<input type="checkbox"/> Encoder <input type="checkbox"/> Microstep	<input checked="" type="checkbox"/> <input type="checkbox"/>
#3	0.00	#3 2	<input type="checkbox"/> Encoder <input type="checkbox"/> Microstep	<input checked="" type="checkbox"/> <input type="checkbox"/>
#4	0.00	#4 4057	<input checked="" type="checkbox"/> Encoder <input type="checkbox"/> DC Servo	<input checked="" type="checkbox"/> <input type="checkbox"/>
#5	0.00	#5 0	<input type="checkbox"/> Encoder <input type="checkbox"/> Microstep	<input checked="" type="checkbox"/> <input type="checkbox"/>
#6	0.00	#6 0	<input type="checkbox"/> Encoder <input type="checkbox"/> Microstep	<input checked="" type="checkbox"/> <input type="checkbox"/>
#7	0.00	#7 0	<input type="checkbox"/> Encoder <input type="checkbox"/> Microstep	<input checked="" type="checkbox"/> <input type="checkbox"/>

Help Close

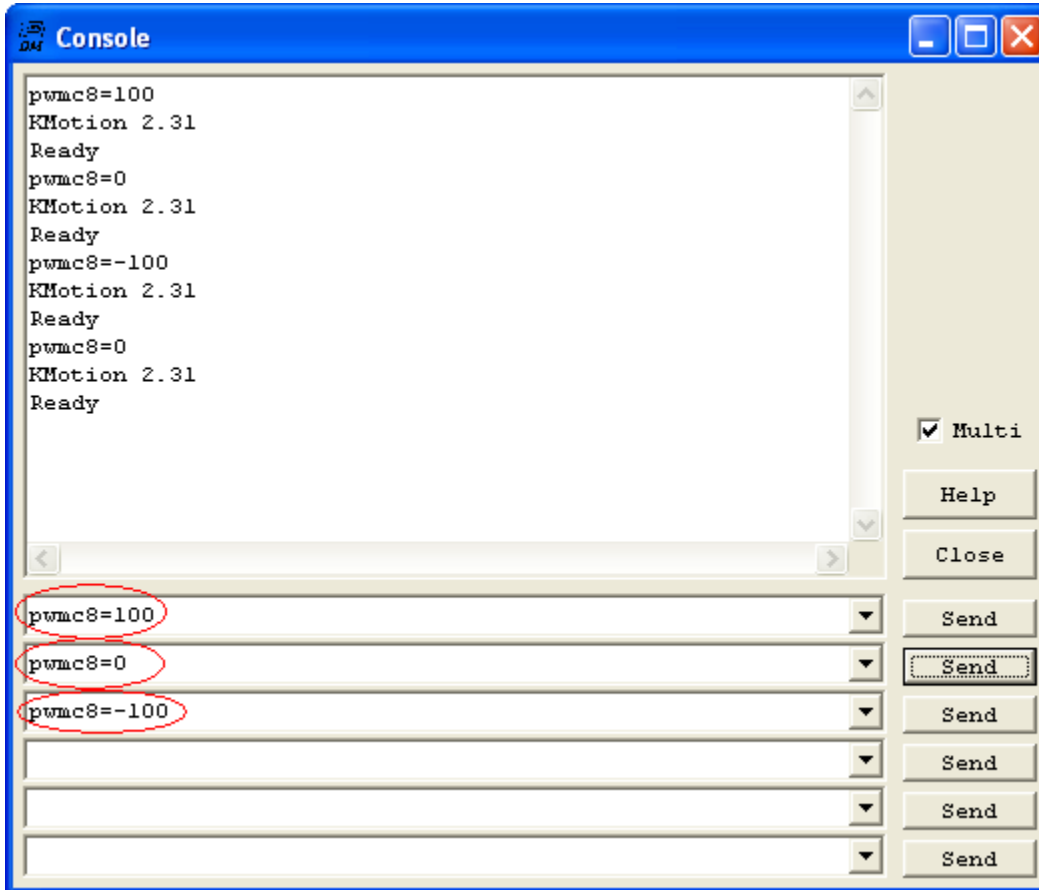
Testing the Motor Amplifier

In order to test the Motor Amplifier functionality Console commands (PWMC - Pulse-Width-Modulation Current Mode) will be entered to drive the motor in the forward and reverse direction. The axis should and must be disabled to issue these commands, otherwise they will be immediately overwritten with servo output commands. Small values should be used initially. Note: PWMC commands may only be used with a SnapAmp (Software Version 2.31 or later), the KMotion onboard amplifiers do not have current feedback mode so PWM commands should be used instead. SnapAmps's PWMC current commands range from -1000 to +1000 which corresponds to approximately -30Amps to +30 Amps. So a value such as 100 might be used to provide a drive of 3 Amps.

Before issuing the PWM commands, SnapAmp's peak current fault levels should be set. After verifying that the axis is disabled (on the Analog Status Screen), load and execute the following program to set the peak current limits. The Fault Green LED on the SnapAmp should turn off. The "I'm alive" Green LED should remain blinking.



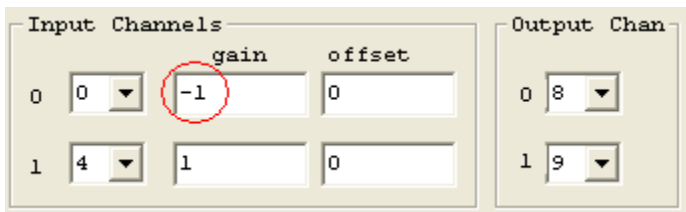
Setup the Console Screen as shown below. Sending the commands should drive the motor in one direction, then no motor drive, then drive in the other direction respectively.



Closing the loop

If the motor and encoder are working we are ready to attempt to close the servo feedback loop. All that is required is to enable the axis by pressing "Enable" on the Step response Screen. Normally one of two things will occur. Either the axis will respond as a weak servo, or the servo will have positive instead of negative feedback and runaway from the target destination rather than toward the target destination.

If the servo has positive instead of negative feedback, something should be reversed. Either the Motor leads may be reversed, the encoder signals may be reversed, or a Input Gain of -1 may be entered on the Configuration Screen (as shown below).

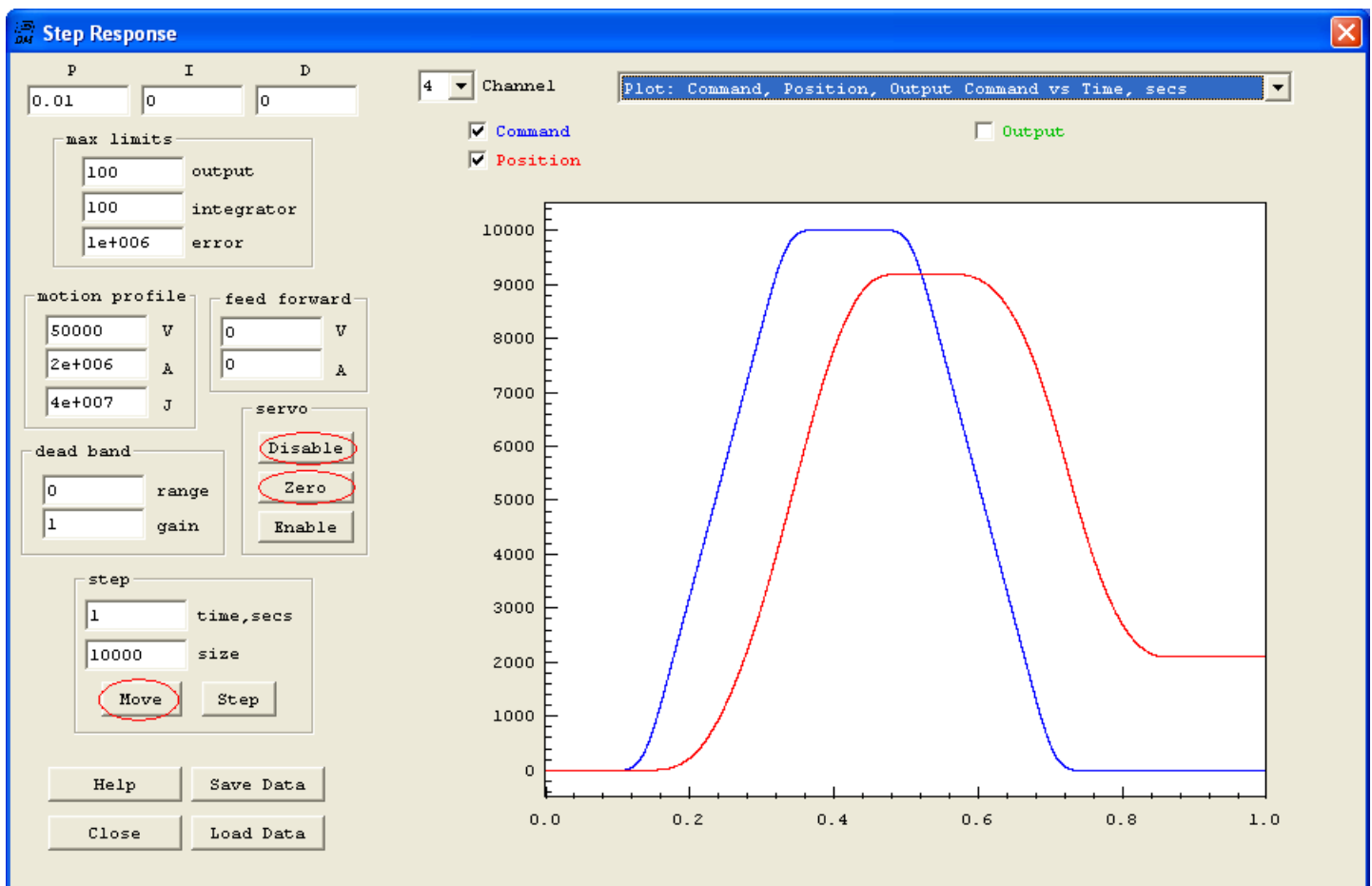


If the servo has proper feedback and responds as a weak servo, as the motor/encoder shaft is turned the servo will cause torque to restore the position back to where it was. With the example proportional gain of 0.01, turning the shaft 1 turn should generate an error of 4000 counts. 4000 counts multiplied by a gain of 0.01 = 40 PWM counts or $40 / 1000 * 30\text{Amps} = 1.2\text{ Amps}$.

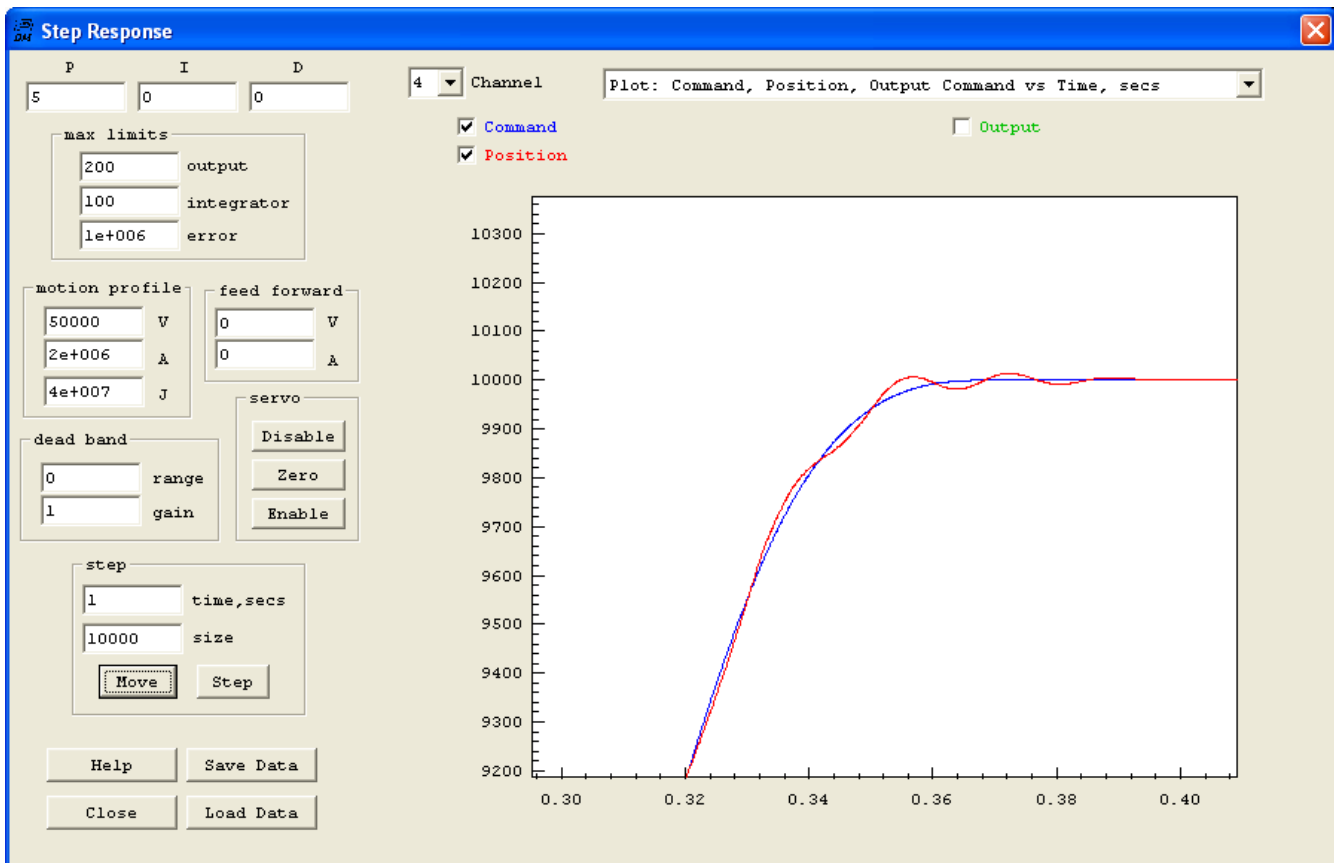
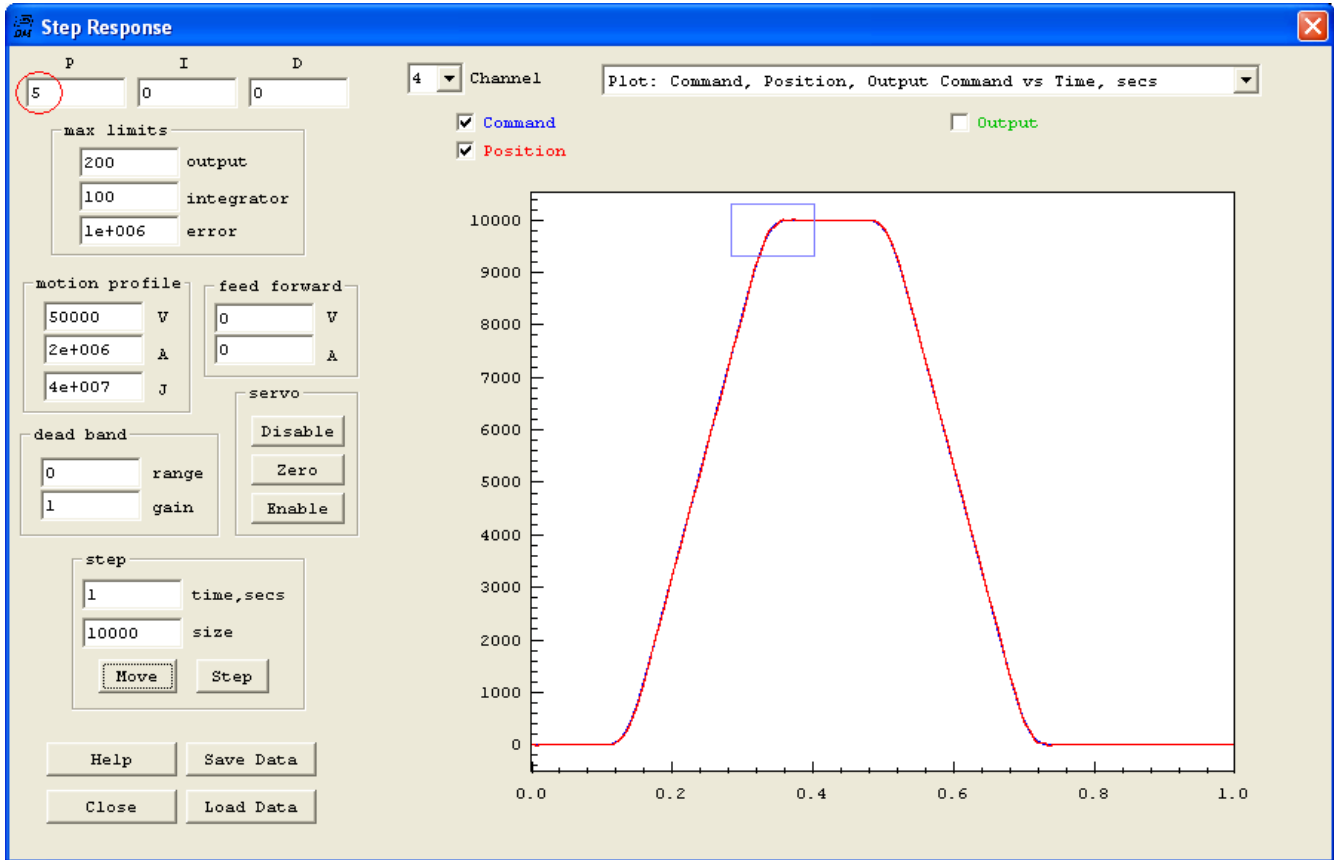
Tuning the Servo

Once the servo loop is functional Servo Tuning should be performed to move the performance, hopefully up to the desired levels. Servo tuning is a complex subject that has been written about extensively. The basic idea is to increase the feedback gains in order to reduce following errors and speed of response without going unstable (oscillating wildly). KMotion has extensive plotting tools to help tune and understand how well a set of parameters performs.

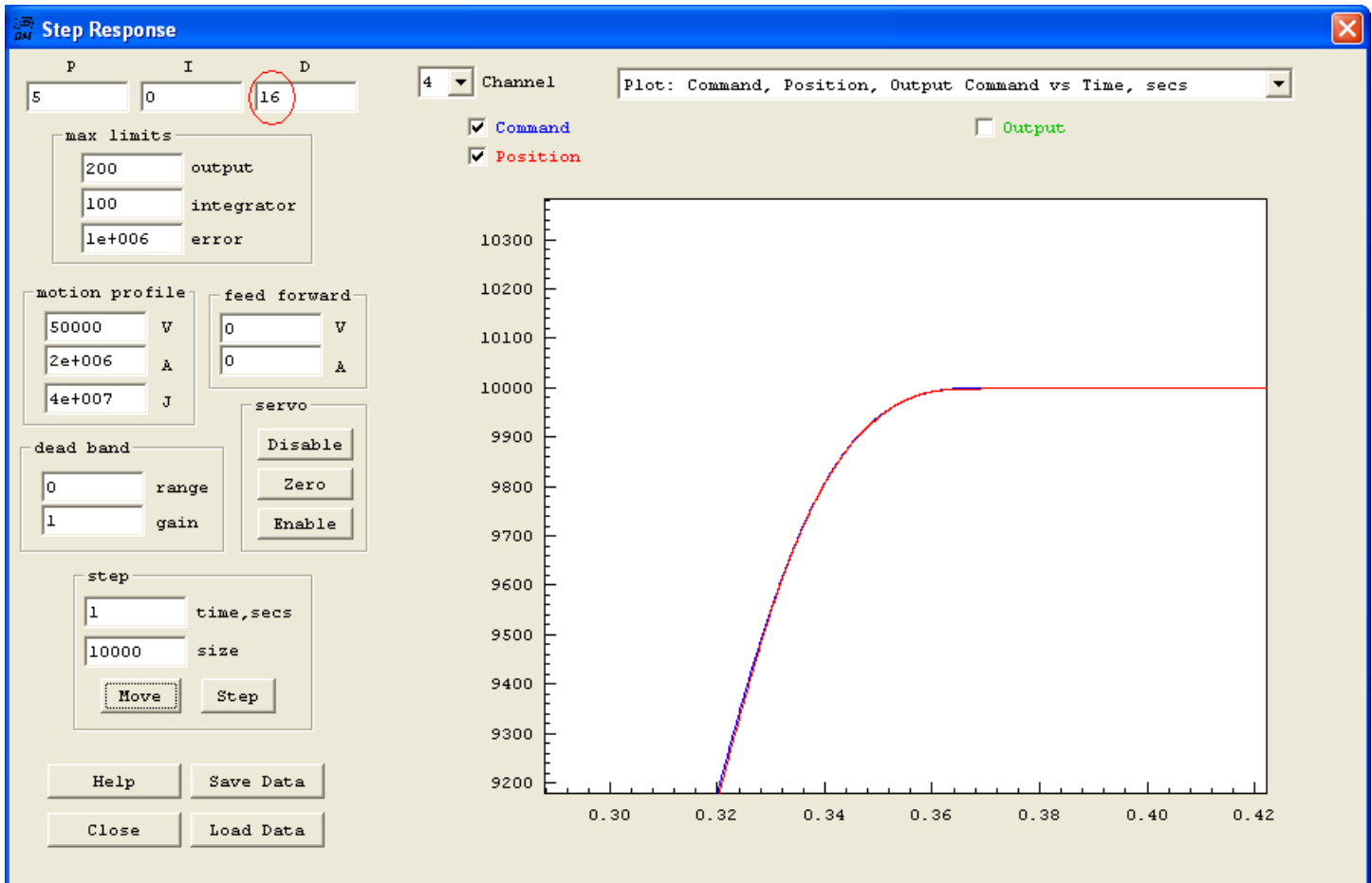
To test the current performance, disable the axis, manually move it to a position where it may move +10000 counts, then push the "Move" button. A plot such as that shown below should be observed. Set the plot type and axis selections to agree with those shown below. The blue plot is the desired path as defined by the motion profile and the motion is made for 10,000 counts and then a second motion is made back to zero. The red plot is the measured encoder position. Note that the red plot attempts to follow the blue plot but with a large error of thousands of counts. Gradually increase the P gain (possibly by factors of 2) while pushing the Move button and the following error should reduce. At some point the system will probably go into oscillation and become unstable. It may be necessary to disable the axis, reduce the gain, and re-enable the axis.



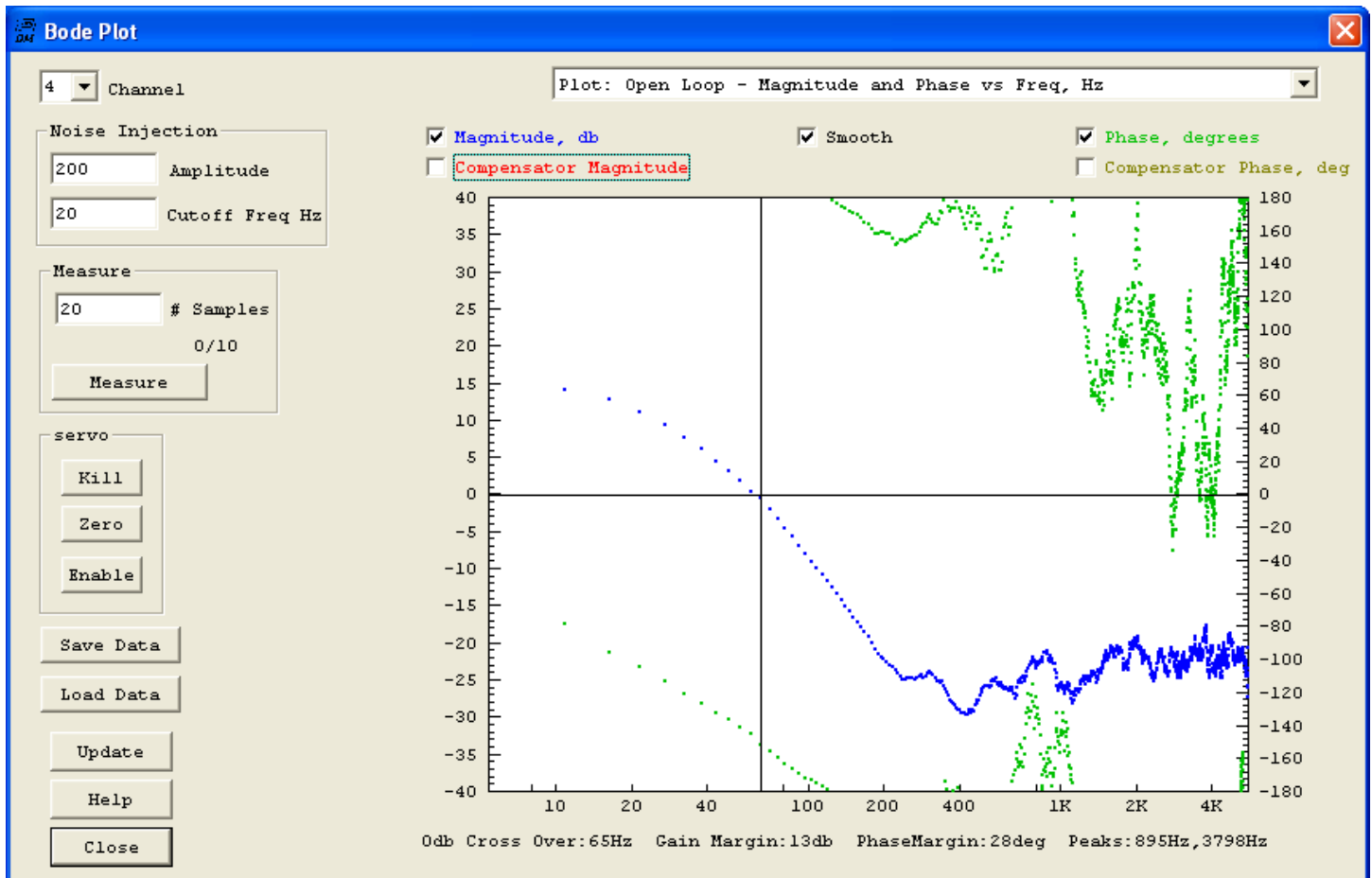
Increasing the gain 500X (from 0.01 to 5) results in a much better response. Notice the blue (desired) plot is almost completely hidden behind the red measured plot. However by zooming in to the purple rectangle (left click mouse drag) we can see oscillation. See the second plot below.



Increasing the derivative gain can often reduce oscillation. In the plot below we have set the D gain to 16. Notice the oscillation has been reduced.

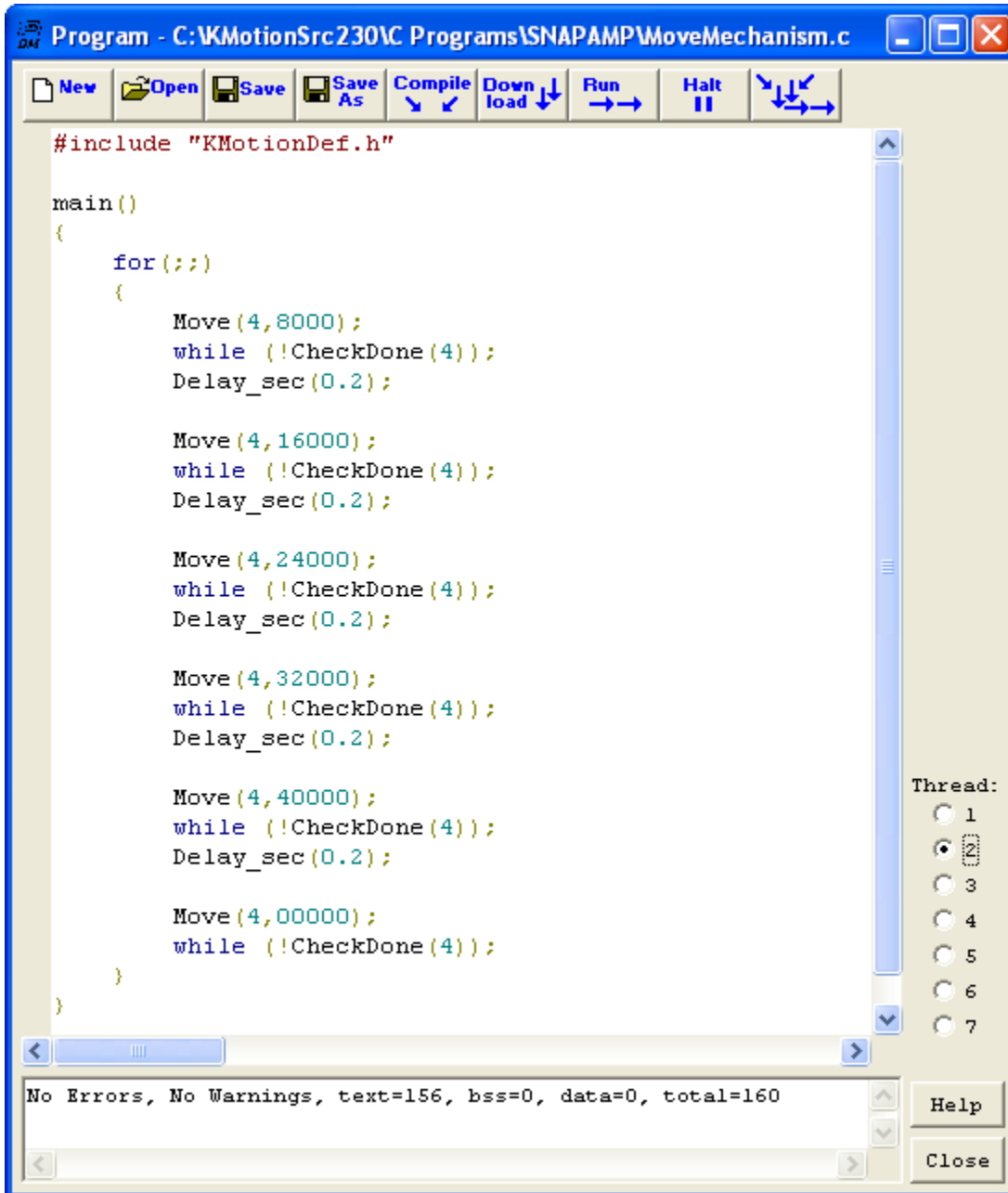


A Bode plot is useful in understanding the frequency response and for designing advanced servo loop filters. Below is the Bode plot response of the system. Important features to look for in a Bode plot is where the Magnitude (blue plot) first passes through the 0 db line. This is the bandwidth of the system. Another important measure is the "phase margin" which is how far away the phase is from -180 degrees when the magnitude is at 0db. In this case the phase margin is 28 degrees. See the Bode Screen in the main Help section for more information



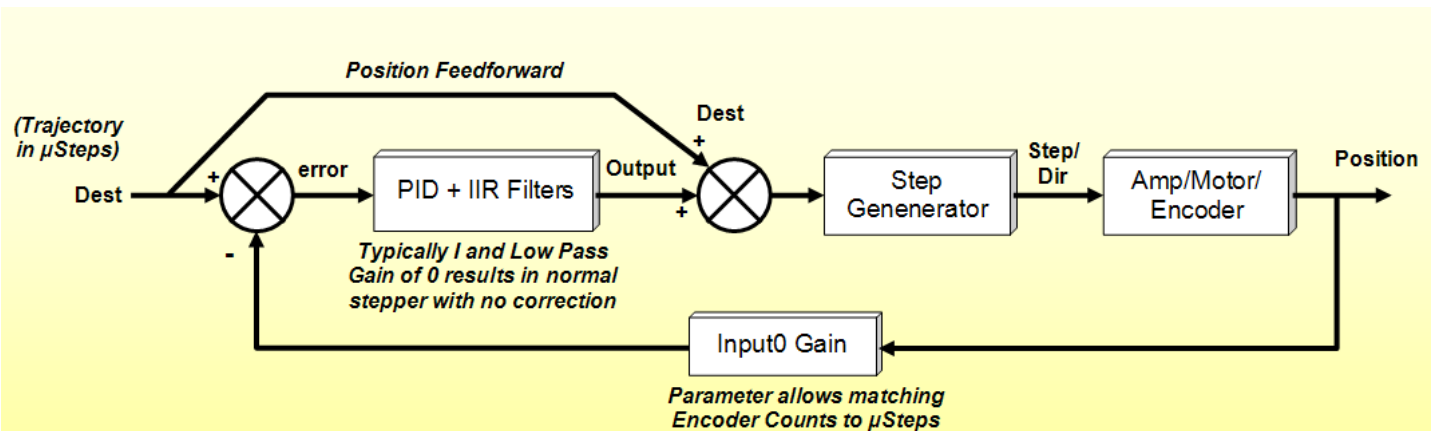
Executing a simple C Program Motion Sequence

The C program below is an example of a simple program to perform a sequence of moves and repeat. There are 5 moves of 8000 counts (2 motor revs) and then a move back to the start. After each small move there is a wait until the motion is complete followed by a 0.2 second delay. [Here](#) is a video of the mechanism executing the motion.



Configuring Closed Loop Step and Direction Outputs

The Closed Loop Stepper Mode works much the same as open loop Step and Direction Output Mode except there is also an error feedback correction term. In fact, if the gains of the correction term (PIDs) are set to zero then the mode will behave the same as an open loop Step Direction Mode. This mode is much easier to “tune” than a stepper driven as a brushless motor and unlike a brushless motor there are no commutation issues. A good application for this mode is a stepper with linear glass scales. The main feature is the position feed forward with fixed gain of 1.0. See the flow diagram below. Without any correction it behaves just like a stepper. As correction gains are added, corrections for drift, friction, load forces, or even a miss step are made. One disadvantage is that the motor can still stall. After the stall and after the motion stops the servo loop could then gradually correct the position which could be of value in some applications.



Closed Loop Stepper Flow Diagram

To Configure an axis a Closed Loop Servo select "CL Step" as the output mode for the axis as shown below. Any input mode may be used as position feedback, but the most common is a quadrature encoder either on the motor shaft or as a linear glass scale.

Note that an Input gain of 1.25 is used in this example. This was calculated from the ratio of the number of μSteps/rev to the number of encoder counts/rev. (A micro stepper amplifier set to 50 microsteps/full setp and a standard stepper motor with 200 full steps/rev will have 10,000 μSteps/rev, a rotary encoder with 2000 lines/rev will profuce 8000 quadrature counts/rev, $10000/8000 = 1.25$). The raw axis units will be in μSteps.

Max Following Error may be used to trip an axis disable when exceeded.

Configuration

Channel 1

Microstepper Amplitude 250

Axis Modes

input Encoder

output CL Step

Max Following Error 500

Inv Dist Per Cycle 1

Lead Compensation 0

Input Channels

	gain	offset
0	4	1.25
1	4	1

Output Chan

0	8
1	8

Limit Switch Options

Negative

☐ Watch Limit

☐ Stop when low

bit no. 0

Positive

☐ Watch Limit

☐ Stop when low

bit no. 0

Action Kill Motor Drive

Flash

User Memory

New Version

Recovery

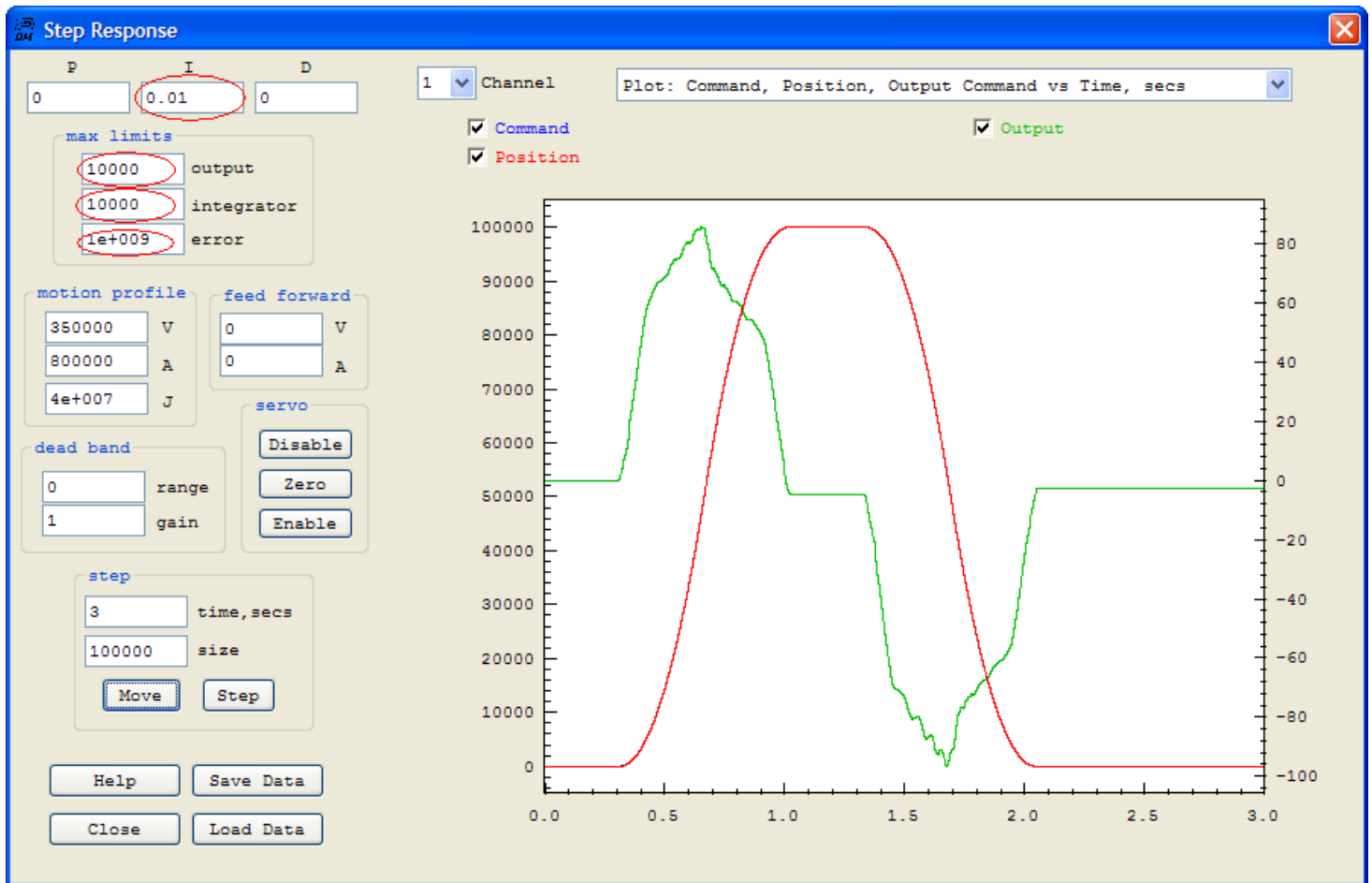
Help

Close

Save Channel Load Channel Download Channel Upload Channel C Code -> Clipboard

Important PID parameters are shown below circled in red. The I (Integrator) gain of 0.01 is probably the most important. Your system may require more or less. Too much and the system may overshoot or become unstable (oscillate). Too little and corrections will be made slowly. Because we are measure the position and also commanding a position Integrator control works well. An integrator will ramp the output at a rate proportional to the amount of the error. This "slowing as we get closer" will result in an exponential curve approaching the target. Backlash, friction, delays, and other factors will eventually cause the system to overshoot and become unstable with too much gain.

Max limits may also be useful for limiting the correction. In this example the limits are set to large values. Limiting the max error to a small value will limit the maximum slew rate of the Integrator. Max Output and Max Integrator are similar for an Integrator only compensator and will limit the maximum amount of correction that can be made.

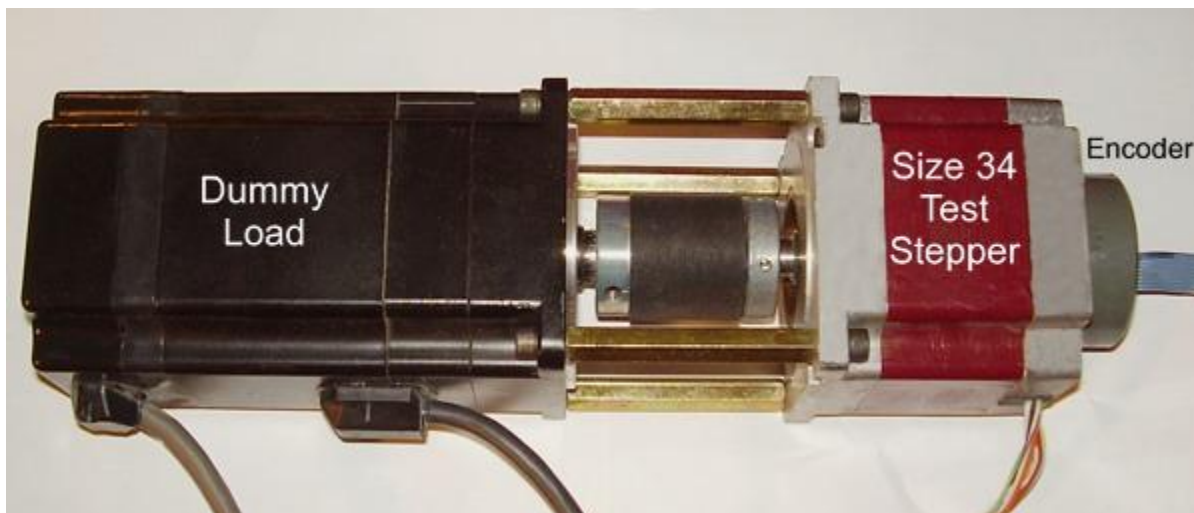


A 2nd order Low Pass Filter is also used in this example to make the system more stable by reducing high frequency corrections. Note a cutoff frequency of 100Hz with Q 1.4 is used. After specifying the filter the Compute Button must be pressed to compute the Z domain IIR Coefficients that are downloaded and used by KFlop.

The screenshot shows the 'Filters' window with three filter configurations. Filter 2 is highlighted with a red circle around the 'Low Pass 2nd' dropdown. The 'Compute' button for Filter 2 is also circled in red. The IIR coefficients for Filter 2 are: B0=0.000768, B1=0.001537, B2=0.000768, A1=1.92081, A2=-0.92388.

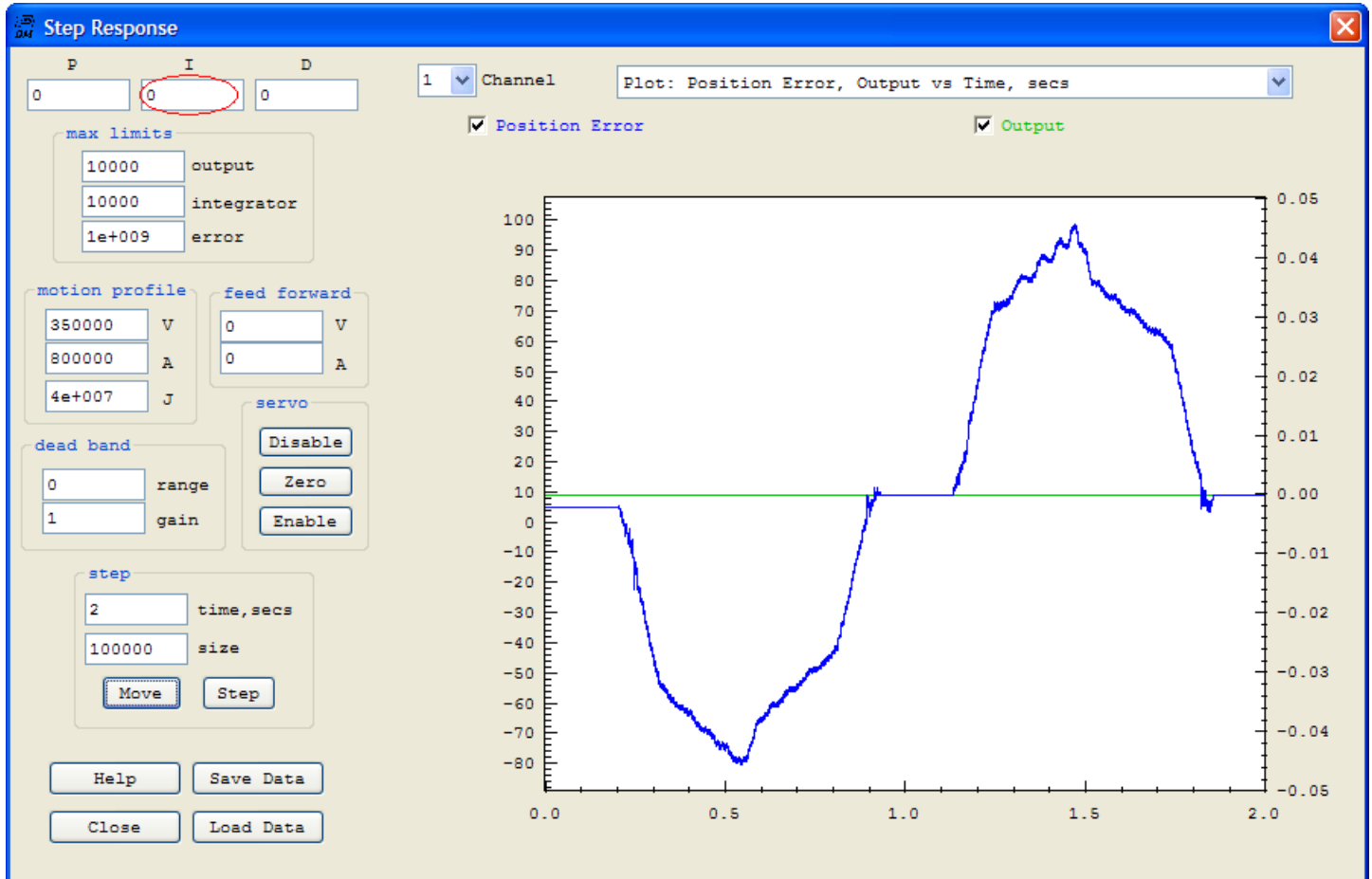
Filter	Domain	Type	Equation	Freq (Hz)	Q	B0	B1	B2	A1	A2
Filter 0	Z Domain		$\frac{y(z)}{x(z)} = \frac{B_0 + B_1 z^{-1} + B_2 z^{-2}}{1 - A_1 z^{-1} - A_2 z^{-2}}$			1	0	0	0	0
Filter 1	Z Domain		$\frac{y(z)}{x(z)} = \frac{B_0 + B_1 z^{-1} + B_2 z^{-2}}{1 - A_1 z^{-1} - A_2 z^{-2}}$			1	0	0	0	0
Filter 2	Z Domain	Low Pass 2nd	$\frac{y(s)}{x(s)} = \frac{1}{s^2 / \omega_n^2 + Qs / \omega_n + 1}$	100	1.4	0.000768	0.001537	0.000768	1.92081	-0.92388

Test Mechanism with Size 34 Stepper with encoder connected to a Dummy Load.



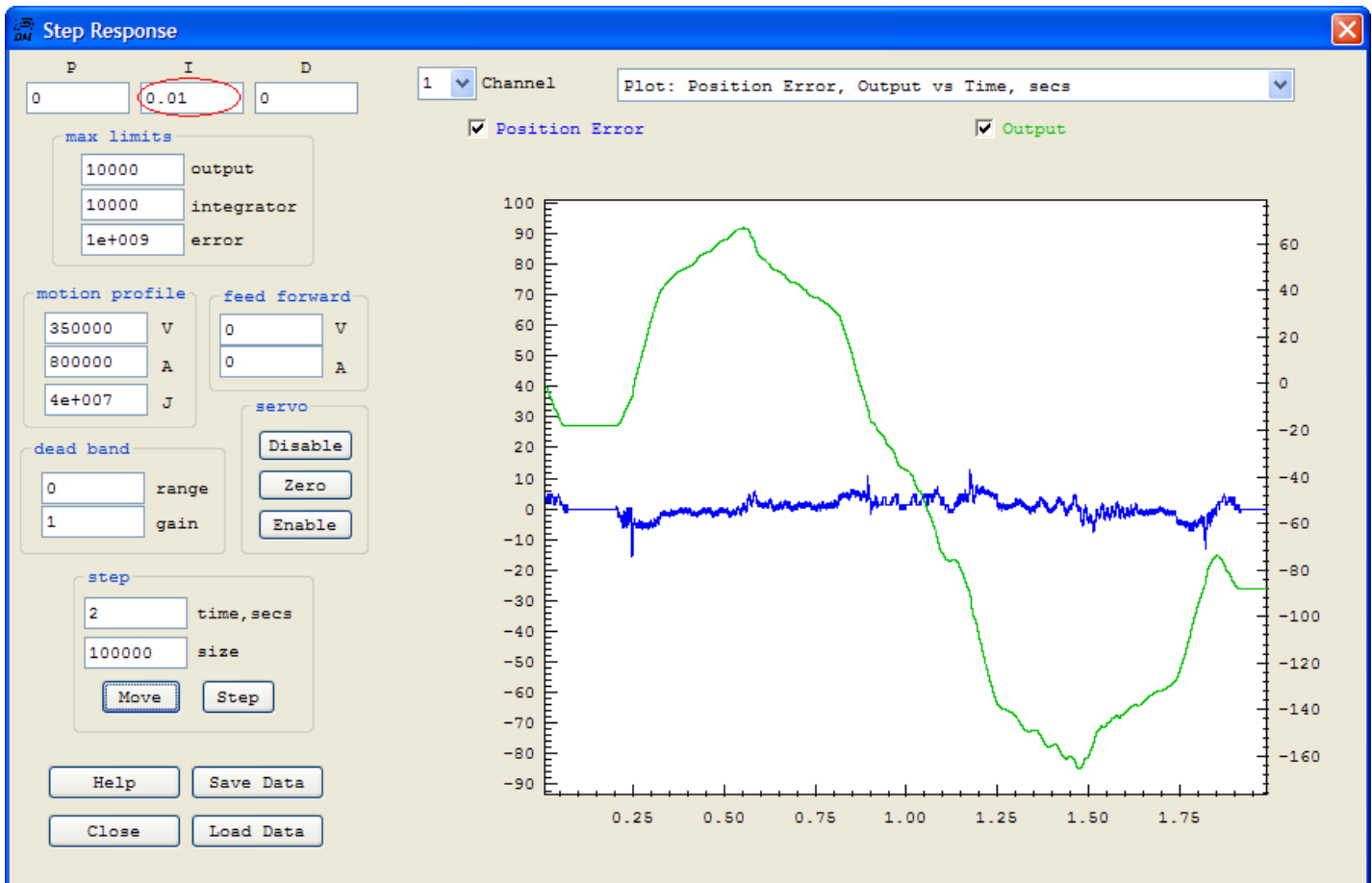
Test Move of a Size34 Stepper with 50 usteps/full step forward and backward at 4000sps

With PID gain zero (no correction). Note encoder shows errors of ~100uSteps



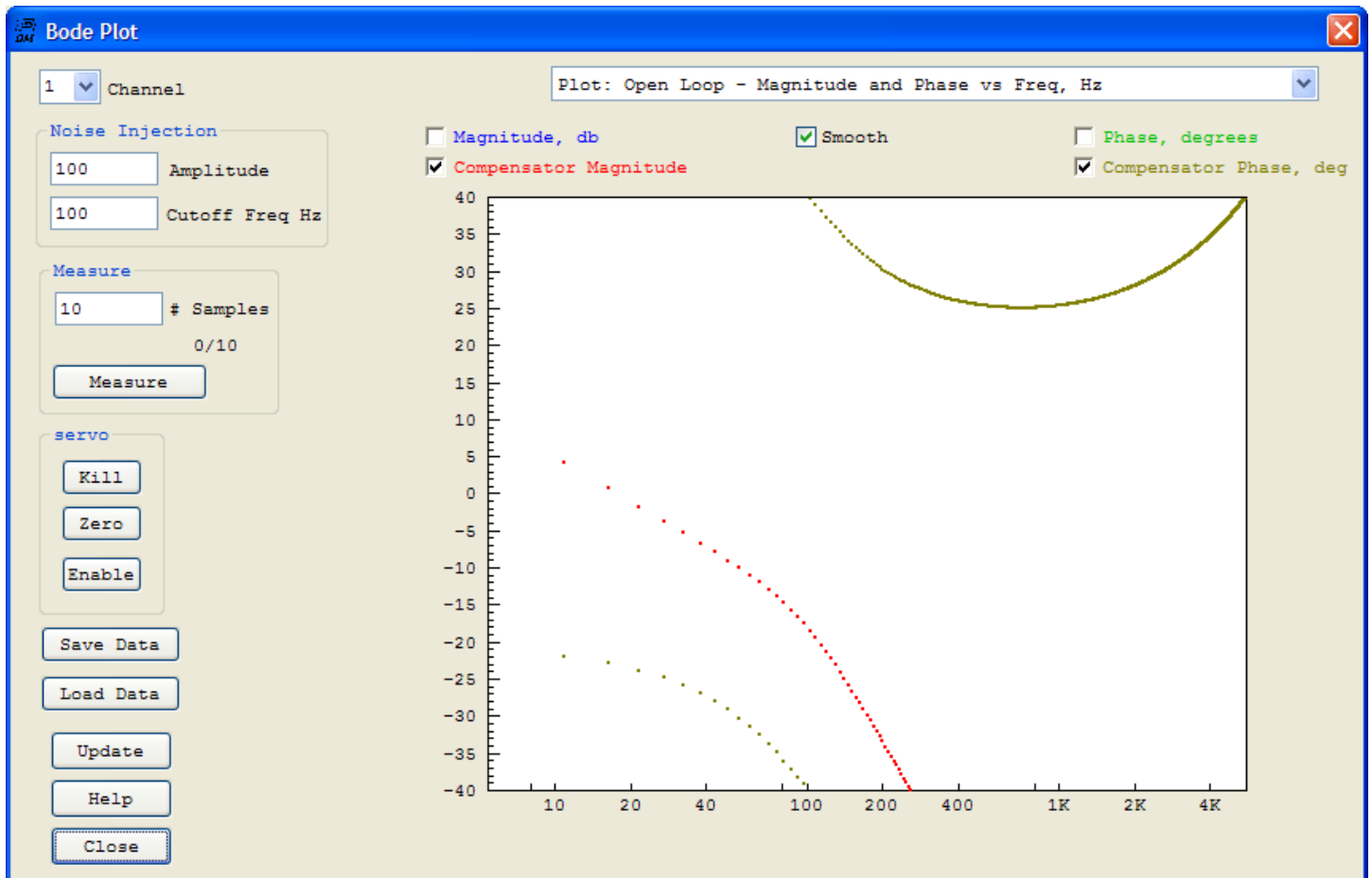
Integrator gain now set at 0.01. Also 2nd Order 100Hz Low Pass Filter Q=1.4 used

Note error is reduced. Blue plot is position error, green is the Output (correction offset)



A Bode Plot of the Compensator PID + LP Filter response. I=0.010 and 2nd order Low Pass 100Hz @ Q=1.4. Red plot is Magnitude.

Note that errors less than about 20 Hz will be corrected. Correction gain drops below 1 (0db) at higher frequencies.



Configuring a Resolver as Input to KMotion

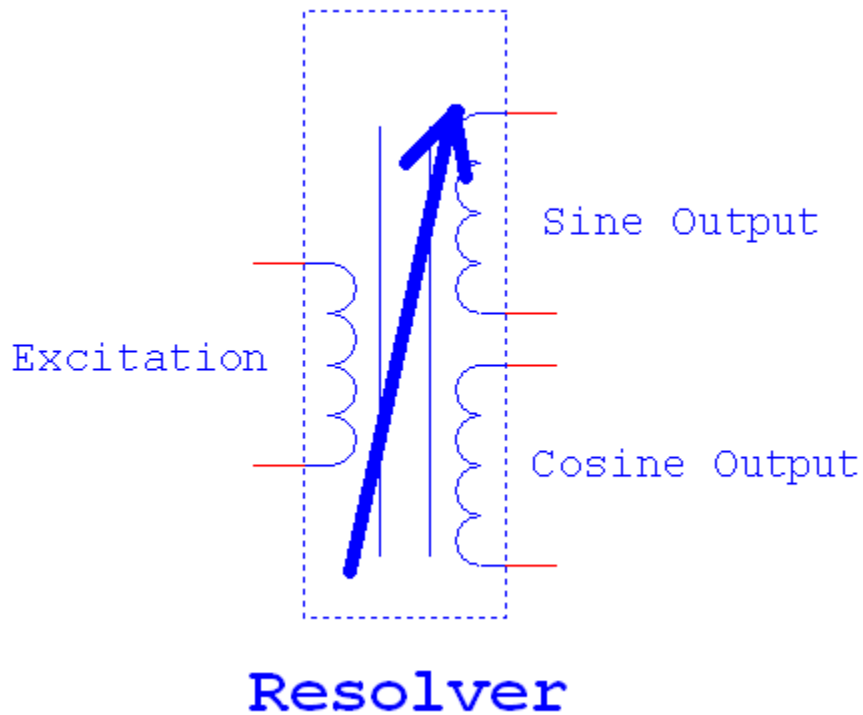
[Here is a video overview.](#)



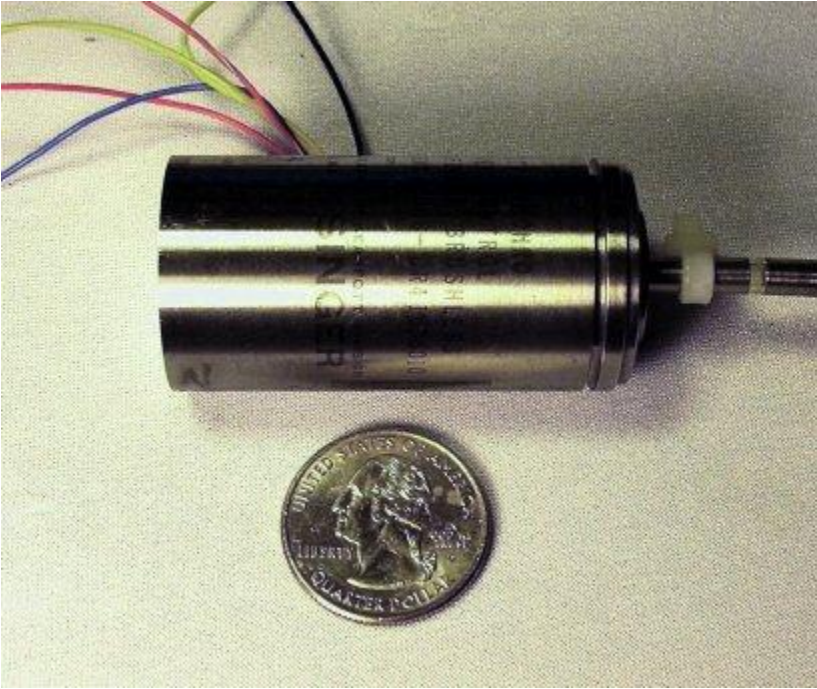
This example configuration shows a Resolver used as a KMotion input device. A Resolver serves a purpose much like a digital encoder. A Resolver is physically somewhat similar to a transformer with two output windings called sine and cosine outputs. For a given excitation input, the amplitude of the sine output is proportional to the sine of the mechanical rotor position, and the amplitude of the cosine output is proportional to the cosine of the mechanical rotor position. By driving the excitation coil of a Resolver and measuring the amplitude of the sine and cosine outputs, the mechanical rotor position can be determined. Although a Resolver is an analog device there are some significant advantages. A resolver is a high reliability device originally developed for military applications. Because a Resolver is able to determine the rotor position in an absolute manner for an entire shaft revolution it is not possible to "lose" position of a fraction of a shaft revolution, as with an encoder. If the position is "lost" it must be lost by a complete multiple of an entire shaft rotation. Often older CNC systems used Resolvers as their feedback device. When using KMotion to retrofit an older system with resolvers, because KMotion may be used with resolvers, it is not necessary to modify the original system to change to digital encoders.

Note: many older systems used a resolver in a somewhat reverse manner. The controller excited the sine and cosine coils in a manner such that third coil would generate a signal proportional to the *difference* between the desired rotor position and the actual rotor position. This difference was then used to directly drive the servo motors to reduce the error. Operating in this manner the controller has no information on the actual rotor position or the following error between the desired and actual rotor positions. A resolver originally used in this manner may still be used in the manner described in this example.

Shown below is a schematic symbol of a Resolver. The arrow indicates a varying coupling between the excitation input and the output coils as a function of rotor angle.



Here is a typical resolver unit (manufactured by Singer) with 6 wires, two for each of the three coils. The wiring of a resolver can usually be determined using a simple ohmmeter. The ohm meter can be used to determine which pair of wires belong to a coil. Furthermore the sine and cosine coils should have a very similar resistance and usually different from the excitation coil. In the resolver shown below the excitation coil has a resistance of 20 ohms and the sine/cosine coils have a resistance of 37 ohms. Swapping the sine and cosine coils will simply reverse the measured motion.



Resolvers may be interfaced to KMotion in one of two ways. One method is where an external circuit or module is available to excite the resolver and determine the sine and cosine magnitudes externally. In this case only the sine and cosine magnitude signals need to be connected to ADC inputs and "Resolver" input mode can be selected. For further information regarding this interface method see the Configuration Screen Setup.

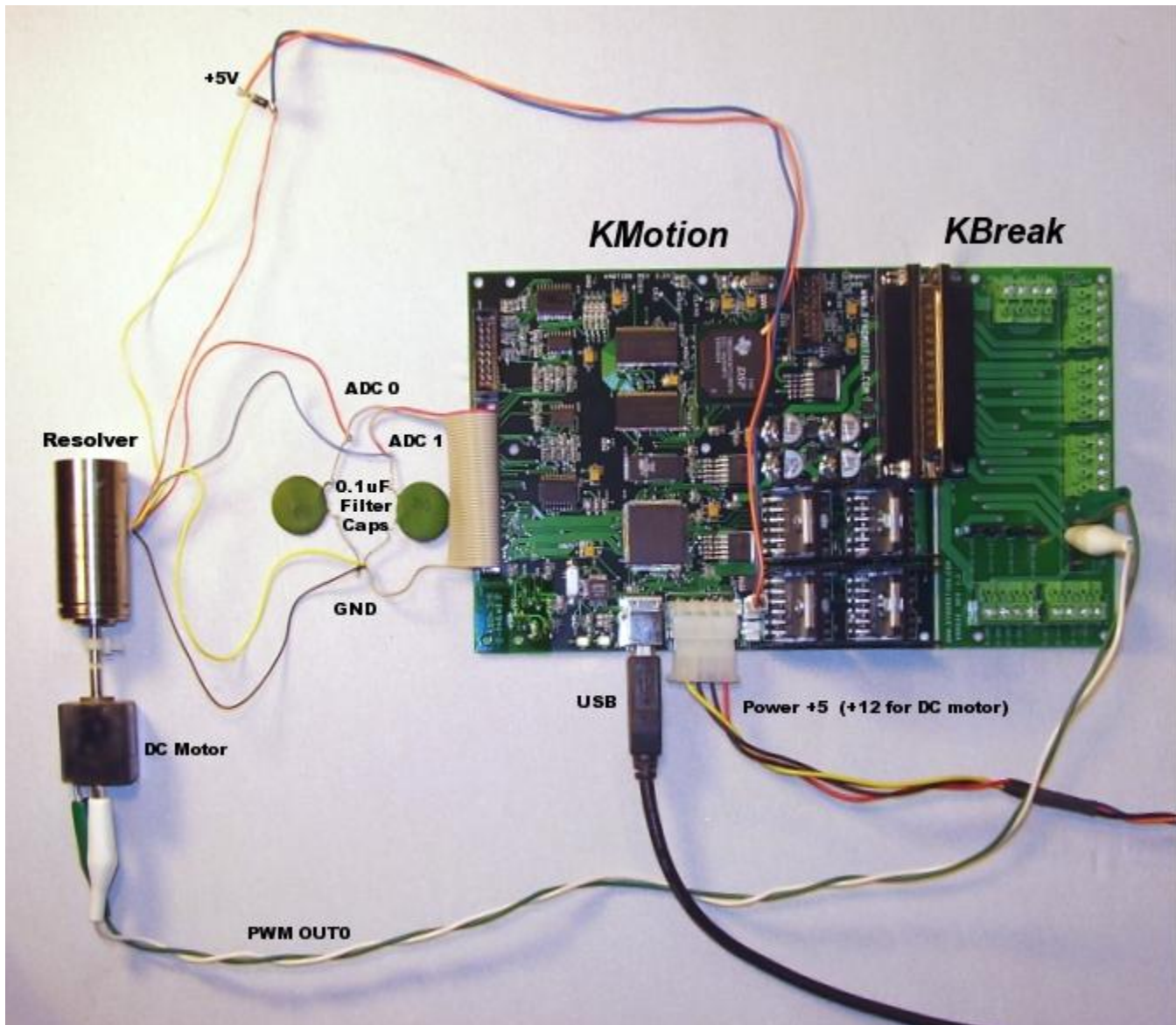
A second method is where KMotion performs the excitation and synchronously samples the AC outputs to determine and track the Resolver position. The second method doesn't require an external Resolver Control Module or control circuit. Only a single diode and two filter capacitors are required. This is the method used in this example configuration. The method does require a KMotion User Program (shown below written in C) to switch the resolver excitation on and off and to sample the output waveforms at the appropriate times to determine the amplitudes.

KMotion's Aux Switch output is used in this example to excite the resolver's input coil with a 5V square wave. +5V is applied to one end of the coil and the Aux Switch drives the opposite end of the coil to ground. This applies +5V to the coil which causes current to ramp up in the coil. When the switch is turned off, current is allowed to recirculate through a diode which will cause the current to ramp down due to the coil's internal resistance. This process is repeated every 4 User Program Time Slices (180us each) or every 720us (1.4KHz). The User Program computes the resolver position on both the positive and negative transitions so the effective update rate is 2.8 KHz. This Aux Switch output, and single diode, may be used to drive several resolvers.

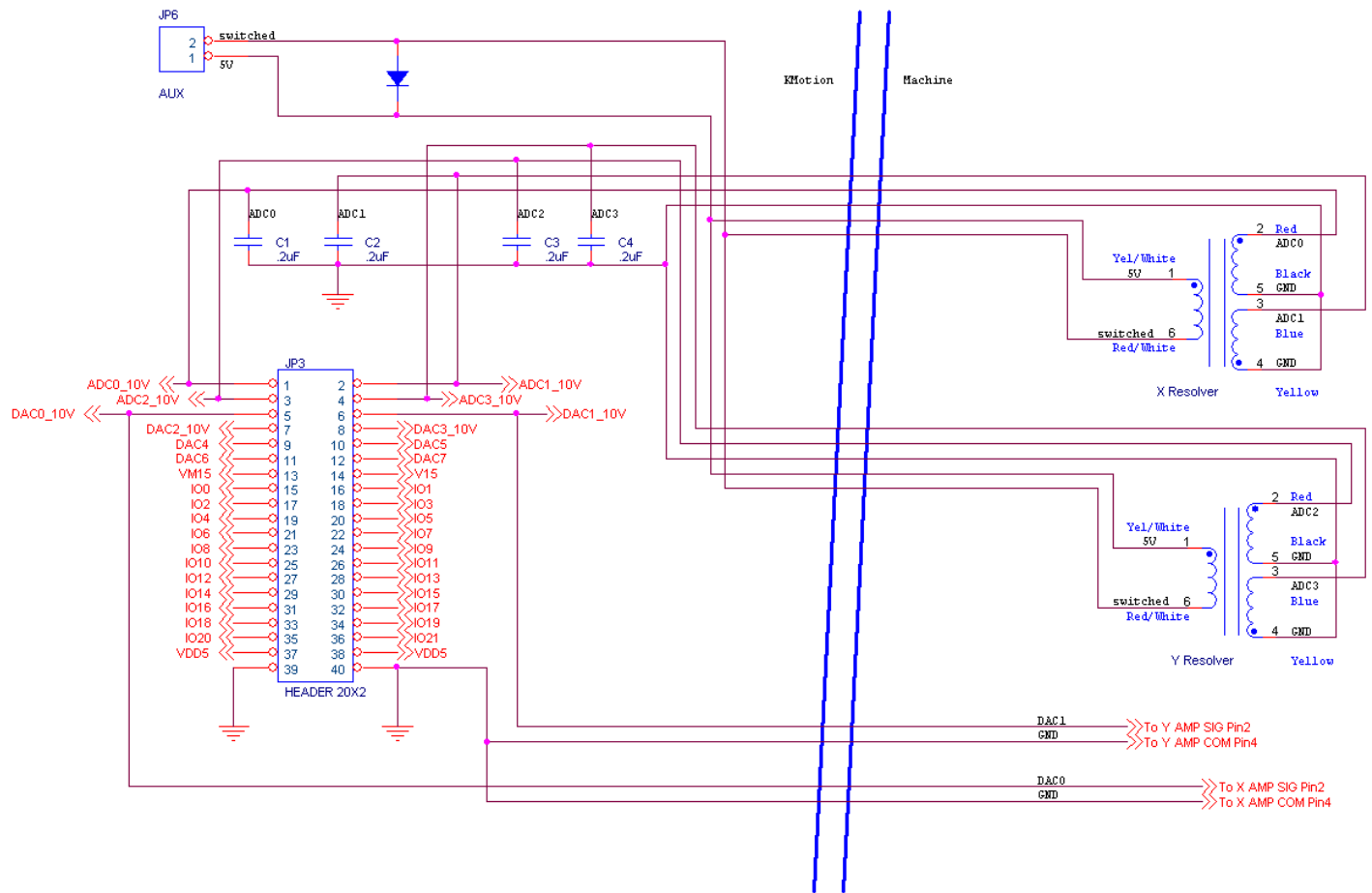
The two output coils of each resolver are each connected to an ADC input with the other ends of the coils connected to ground. 0.2uF filter capacitors are used to smooth the output waveforms and should be located near the KMotion Board.

The example layout below shows the connections for a single resolver. Because KMotion has only 4 ADC inputs, KMotion is limited to interfacing to two resolvers. If more axes are required then digital

encoders must be used. For testing purposes a small DC torque motor is shown connected to the resolver. The DC motor is driven from one PWM output using the +12V supply. A more likely scenario for an actual CNC retrofit would be to use the existing motor power amplifiers driven by KMotion's +/-10V DAC outputs.



A wiring diagram for a complete 2 axis resolver is shown below. Also shown is the DAC output connections for connecting to external power amplifiers with +/-10 V Inputs.



This is a KMotion User Program which basically loops every 4 time slices (720us), switches the 5V excitation, samples the ADC readings, computes the output magnitudes, and calls an internal function (DoResolverInput2) that computes the rotor angle and updates the axis's Position. DoResolverInput2 internally multiplies the measured angle (in Radians) by $1000/(2\pi)$ so that 1 shaft revolution will be seen as 1000.0 counts. This causes numeric values to have similar ranges as with digital encoders.

```
#include "KMotionDef.h"

// Two Axis Resolver Program 1/6/08
//
// outputs square wave to both resolvers using Aux Switch Output
//
// then samples output coils near positive and negative peaks
// takes the difference to compute magnitudes
//
// these ratios are used to match the amplitudes of sine:cosine

#define RATIO0 (978.0f/768.0f) // size j/size k
#define RATIO1 (950.0f/709.0f) // size n/size m
```

```

main()
{
    int i=0;
    int k0,j0,k1,j1;
    int m0,n0,m1,n1;

    SetBit(28); //+/-15V on
    SetBitDirection(30,1); // configure AUX switch as output
    DefineCoordSystem(0,1,-1,-1); // Define 2 axis system

    Delay_sec(0.1); // wait for +/- 15V to be stable
    for (;;) // repeat forever
    {
        WaitNextTimeSlice(); // wait a few servo cycles
        WaitNextTimeSlice();
        Delay_sec(10e-6); // wait for ADC conversion to complete
        k0=ADC(0); // Sample all the ADCs
        j0=ADC(1);
        m0=ADC(2);
        n0=ADC(3);
        SetBit(30); // Switch the resolver excitation

        //compute & track position based on measured magnitudes
        DoResolverInput2(ch0, (k1-k0)*RATIO0,j1-j0);
        DoResolverInput2(ch1, (m1-m0)*RATIO1,n1-n0);

        WaitNextTimeSlice(); // wait a few servo cycles
        WaitNextTimeSlice();
        Delay_sec(10e-6); // wait for ADC conversion to complete
        k1=ADC(0); // Sample all the ADCs
        j1=ADC(1);
        m1=ADC(2);
        n1=ADC(3);
        ClearBit(30); // Switch the resolver excitation

        //compute & track position based on measured magnitudes
        DoResolverInput2(ch0, (k1-k0)*RATIO0,j1-j0);
        DoResolverInput2(ch1, (m1-m0)*RATIO1,n1-n0);

        #if 0 // enable this to print the magnitudes occasionally
            if (++i==1000)
            {
                i=0;
                printf("%5.0f %5d\n",k1-k0,j1-j0);
            }
        #endif
    }
}

```

The Configuration Screen input mode is shown here set to "User Input". This allows the User Program shown above to be in charge of setting the current position.

Configuration

Channel

Microstepper Amplitude

Axis Modes
input
User Input
output
DC Servo

Max Following Error

Inv Dist Per Cycle

Lead Compensation

Input Channels

	gain	offset
0	<input type="text" value="0"/>	<input type="text" value="1"/>
1	<input type="text" value="0"/>	<input type="text" value="1"/>

Output Chan

0	<input type="text" value="0"/>
1	<input type="text" value="1"/>

Limit Switch Options

Negative

☐ Watch Limit

☐ Stop when low

bit no.

Positive

☐ Watch Limit

☐ Stop when low

bit no.

Action

Launch on Power Up

☐ Thread 1

☐ Thread 2

☐ Thread 3

☐ Thread 4

☐ Thread 5

☐ Thread 6

☐ Thread 7

Flash

User Memory

New Version

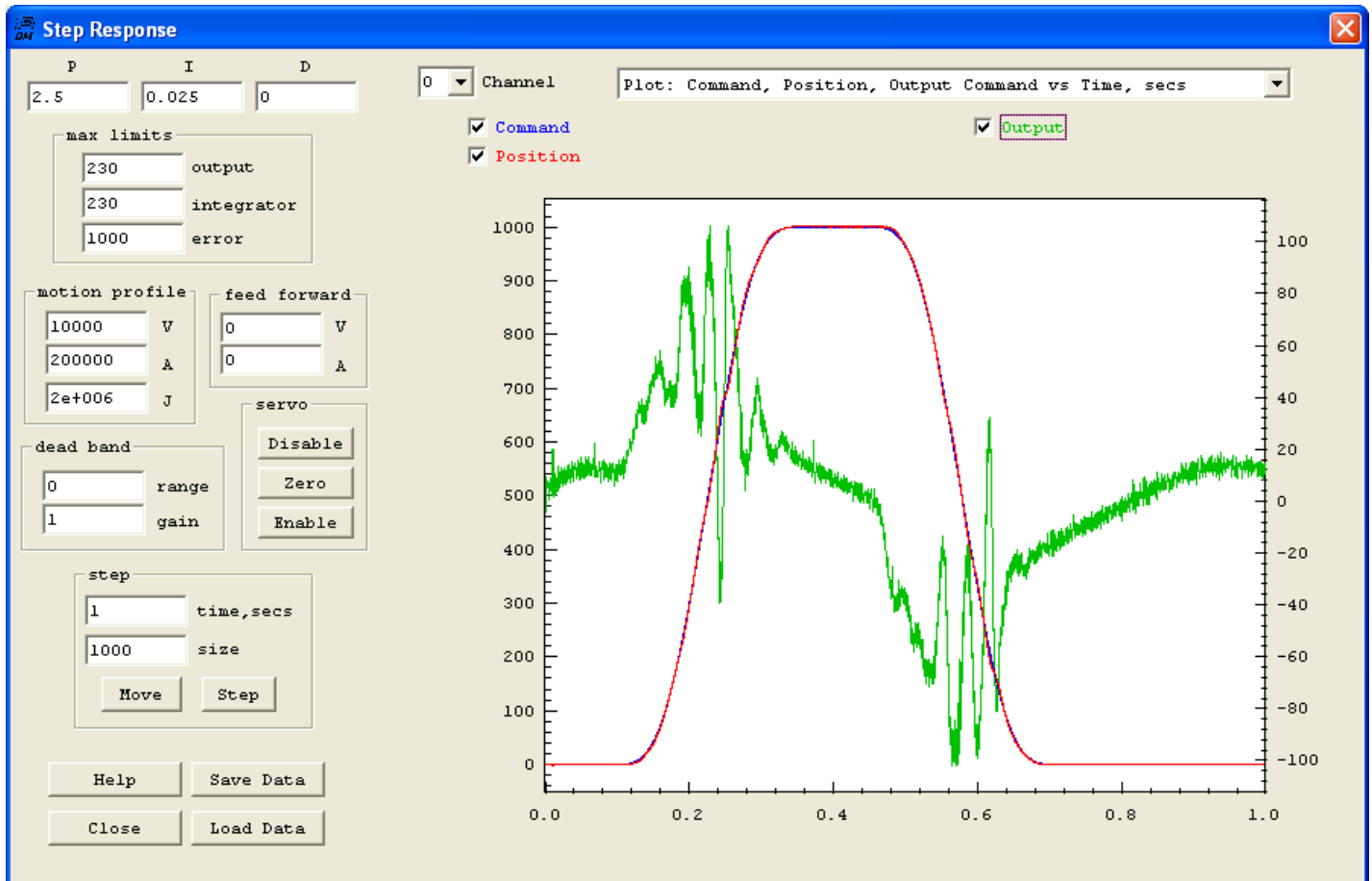
Recovery

Help

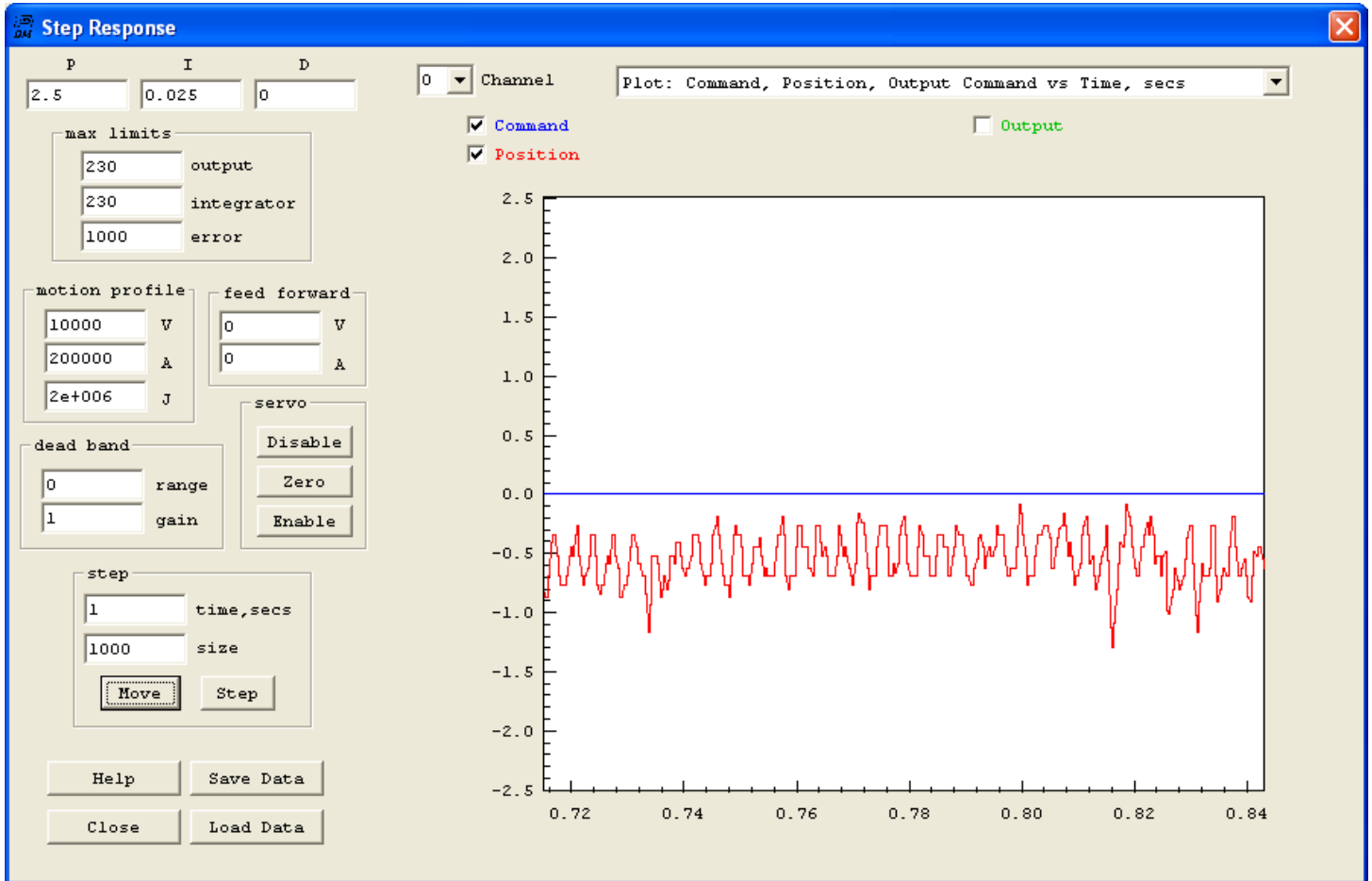
Close

Save Channel Load Channel Download Channel Upload Channel C Code -> Clipboard

Here is a Move Plot of a move of 1000 which corresponds to 1 shaft revolution of the encoder. Not the Command (Blue) and Measured (Red) positions are nearly overlaid.



Zooming in on the position gives an indication of the position resolution and noise level. The signal shown below has a range of approximately ± 0.5 counts where 1 count is $1/1000$ of a shaft resolution. So for example on a system with a 5 pitch lead screw (0.200 inches / rev) this would correspond to ± 0.1 mils ($\sim 2.5\mu\text{m}$).



Data Gathering

KMotion provides a flexible method for capturing data of all types every servo sample period (90µs). This same method is how **KMotion** gathers step response and Bode plot data.

Basically a list of addresses and data types are defined. An end address of where to stop capturing data is set, and when triggered the Servo Interrupt will capture the specified data values. All values are converted to double precision numbers before being placed into the gather buffer. The maximum size of the Gather Buffer is 1,000,000 double precision values (8 MBytes).

```
#define MAX_GATHER_DATA 1000000 // Size of gather buffer (number of doubles, 8 bytes each).
```

The following example shows how to setup to capture the two PWM drives (for a stepper motor) and the commanded destination for a 0.5 second time period, trigger the capture, make a simple move, wait until the capture is complete, and print the results.

```
#include "KMotionDef.h"

main()
{

    int i,n_Samples = 0.5 / TIMEBASE;

    gather.Inject = FALSE; // Don't inject any Data anywhere

    gather.list[0].type = GATHER_LASTPWM_TYPE; // Gather PWM 0
    gather.list[0].addr = &LastPWM[0];

    gather.list[1].type = GATHER_LASTPWM_TYPE; // Gather PWM 1
    gather.list[1].addr = &LastPWM[1];

    gather.list[2].type = GATHER_DOUBLE_TYPE; // Gather Dest axis 0
    gather.list[2].addr = &chan[0].Dest;

    gather.list[3].type = GATHER_END_TYPE;

    gather.bufptr = (double *)0xffffffffc; // force more than endbuf
    gather.endptr = gather_buffer + 3 * n_Samples;

    TriggerGather(); // start capturing data

    MoveRel(0,20); // Start a motion

    while (!CheckDoneGather()) ; // wait till all captured

    // print all captured data (every 50th sample)

    for (i=0; i<n_Samples; i+=10)
```

```
printf("%d,%f,%f,%f\n", i,gather_buffer[i*3],
gather_buffer[i*3+1],
gather_buffer[i*3+2]);

}
```

Data will be printed to the **KMotion** Console Screen which is also written to a permanent log file at:

<KMotionInstallDir>\KMotion\Data\LogFile.txt

Normally data scrolls off of the Console Screen into the permanent log file, to flush all data into the log file, exit the **KMotion** application.

An Excel plot of the captured data is shown below.

