dynomotion.com

Table of Contents

KMotion/KFLOP Executive Software Screens

Kogna Board specific

KFLOP Board specific

General

KMotionCNC

Mach 3

KStep

Kanalog

Konnect

SnapAmp

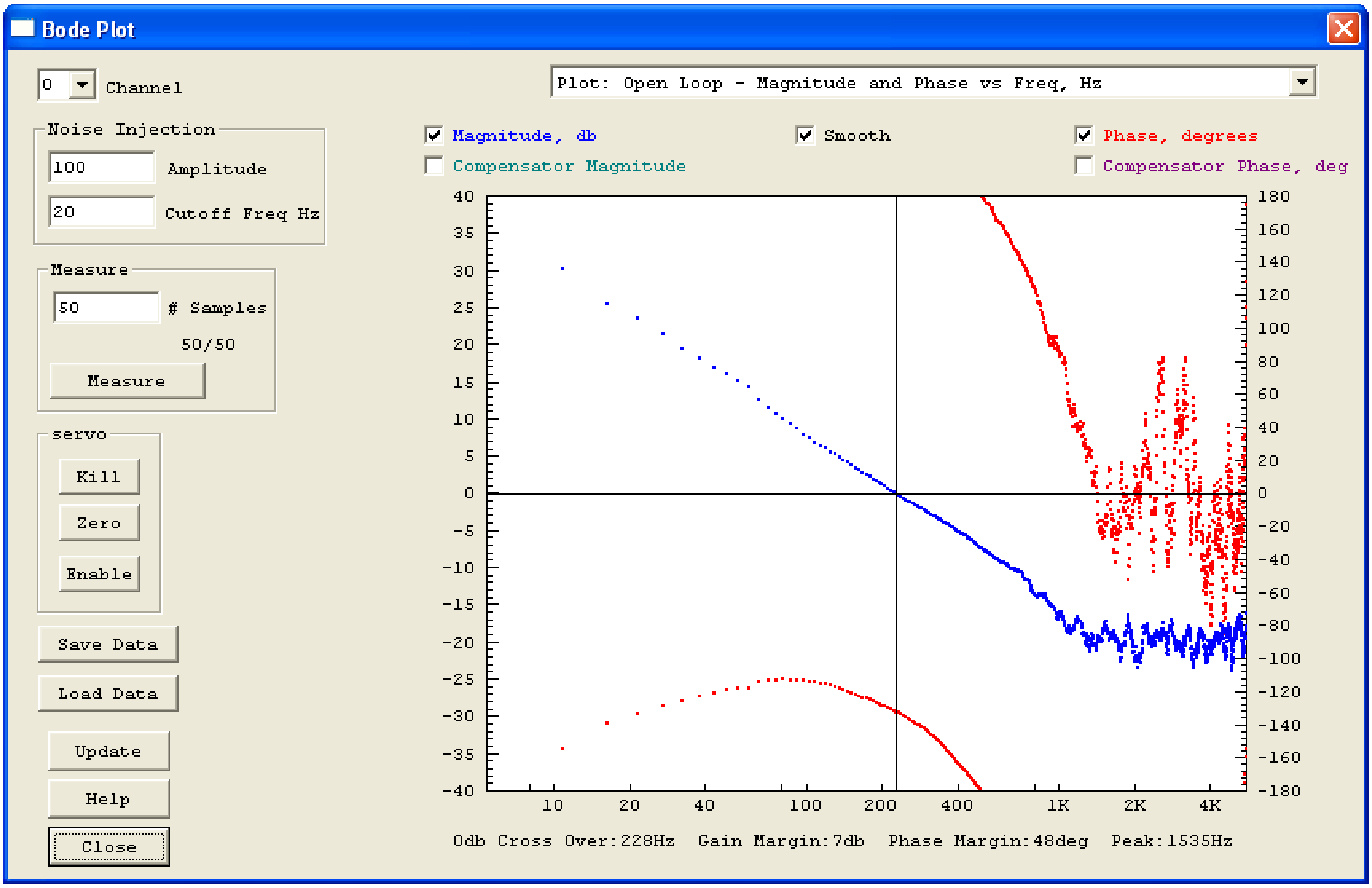
KNozz

Examples

Videos

Forum/Support

Bode Plot Screen



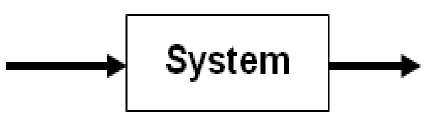
The ***Bode Plot Screen*** allows the measurement of a servo loop and graphs the open loop response. A Bode Plot is by far the most common means used to measure and understand the behavior of a control loop. ***KMotion*** contains advanced built-in features to allow rapid Bode Plot measurement and display. 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.

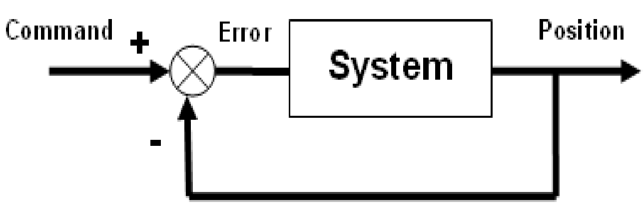
A Bode plot is a useful tool used to examine the stability of a servo feedback loop. If a system has an open loop gain of -1 (magnitude of 0 db and phase of -180 degrees), then if it is placed into a negative feedback loop, it will become unstable and oscillate. Because a system's gain and phase vary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

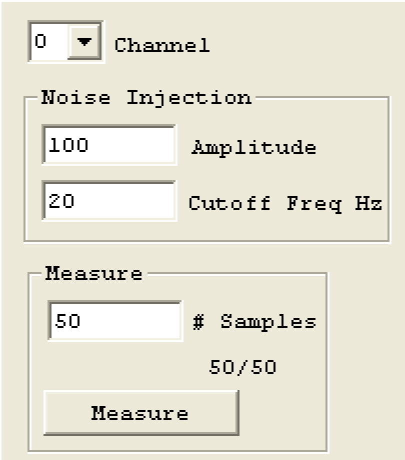


Open Loop



Closed Loop

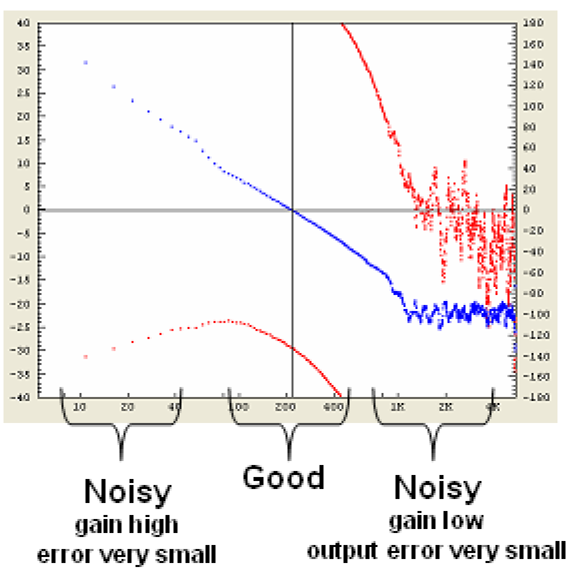
The most direct method for obtaining the open loop response is to break the servo loop open, inject a signal into the system and measure the output. However, this is usually impractical as most systems will run out of their linear range if driven in an open loop manner. ***KMotion*** operates the servo 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.

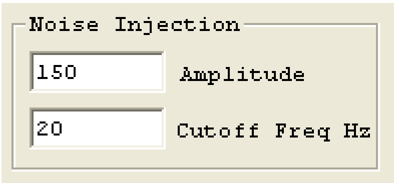
Depending on the type of plot requested (either Time Domain or Frequency Domain) either the last acquired time domain measurement will be displayed or the *average* all the frequency domain measurement so far acquired will be displayed.



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.

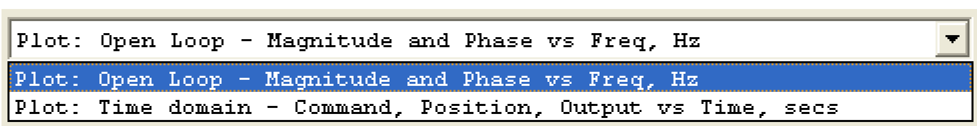
Additionally, instead of injecting sine waves at various frequencies individually, ***KMotion*** uses a technique where a random noise signal that is rich in *many* frequencies is injected. Using a method involving an FFT (Fast Fourier Transform) of the input and output, the entire frequency response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

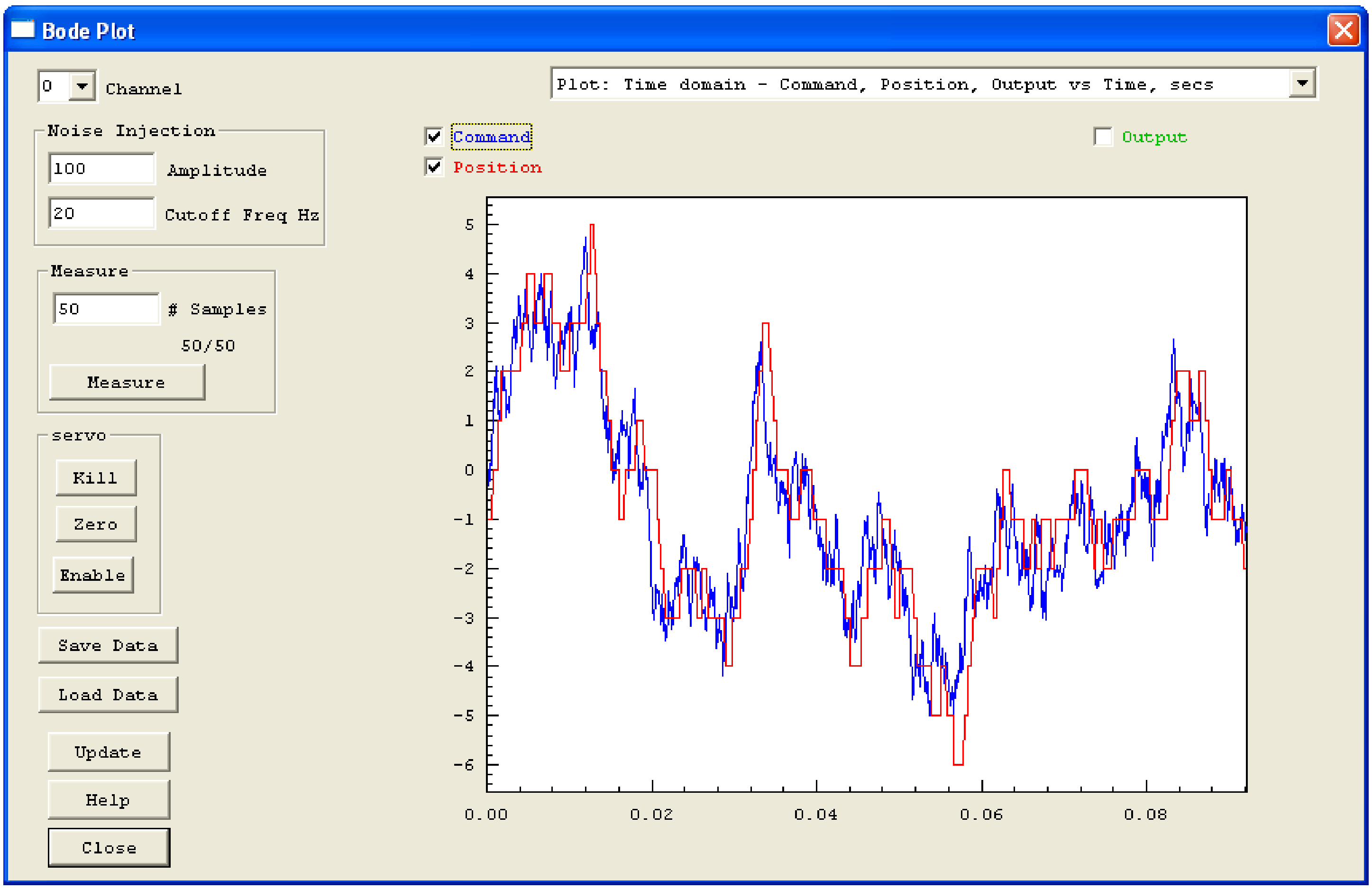


It is therefore very important to verify that while the system is being measured that it is operating in its linear range. This usually entails that the system isn't being driven too hard (or too fast), so that the drive to the system (*Output*) is not reaching saturation. Additionally, it is important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

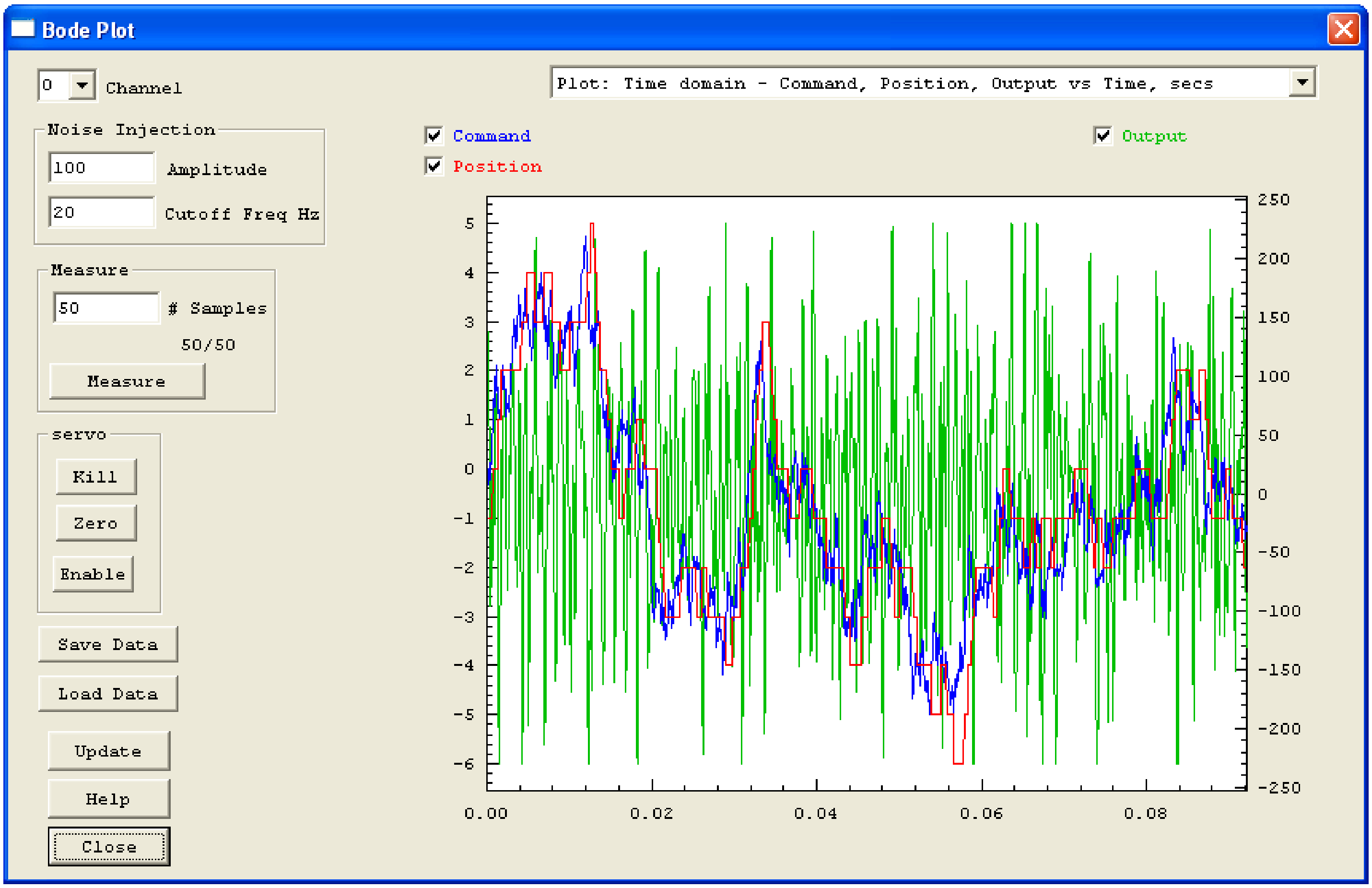


A typical time domain measurement is shown below. The *blue* graph shows the random stimulus to the system. The *red* graph shows the system's response, which in this example is the output of an encoder. Note that the position is quantized to integer counts and has a range of approximately 10 counts. 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.

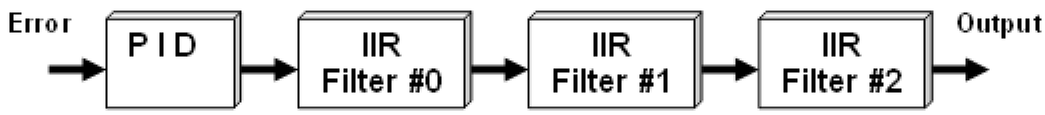


The graphs below include the output drive signal shown in *green*. Note that the output drive is on the verge of saturating at the maximum allowed output value (example is from a 3 phase brushless motor, maximum allowed PWM counts of 230). This shows that we are driving the system as hard as possible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

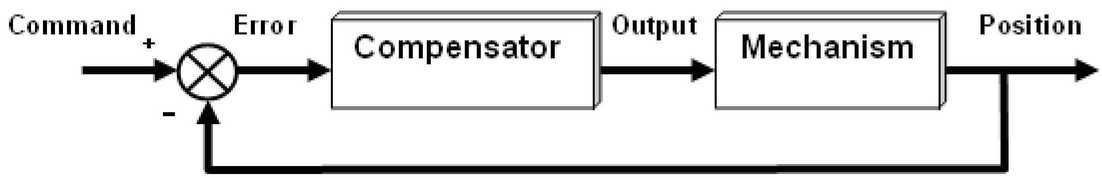
Another means of improving the measurement accuracy (as well as servo performance in general) is obviously to increase the encoder resolution if economically or otherwise feasible.



Compensator Response

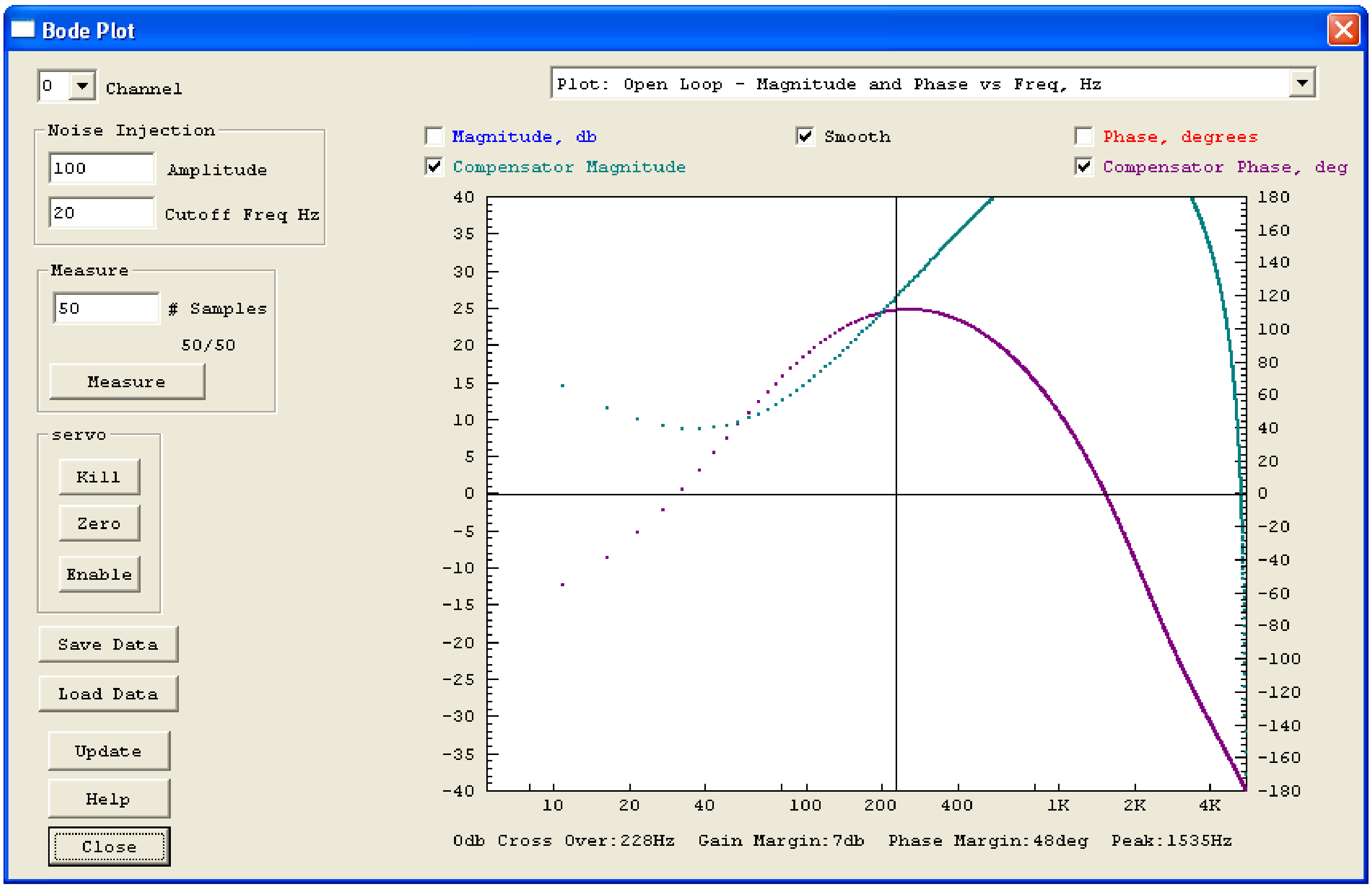


KMotion Compensator

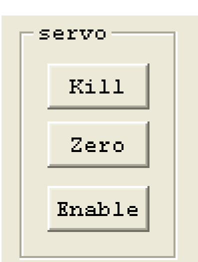


Complete Closed Loop Servo

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.



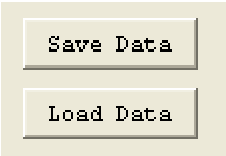
Axis Control



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



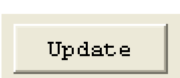
The Save/Load Data buttons allow the captured Bode Plot to be saved to a text file and re-loaded at a later time. The text file format also allows the data to be imported into some other program for display or analysis. The file format consists of one line of header followed by one line of 9 comma separated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

G17 Arcs in XY plane
G18 Arcs in XZ plane
G19 Arcs in YZ plane

G20 Inch units
G21 mm units

G28 Move to Reference Position #1
G30 Move to Reference Position #2

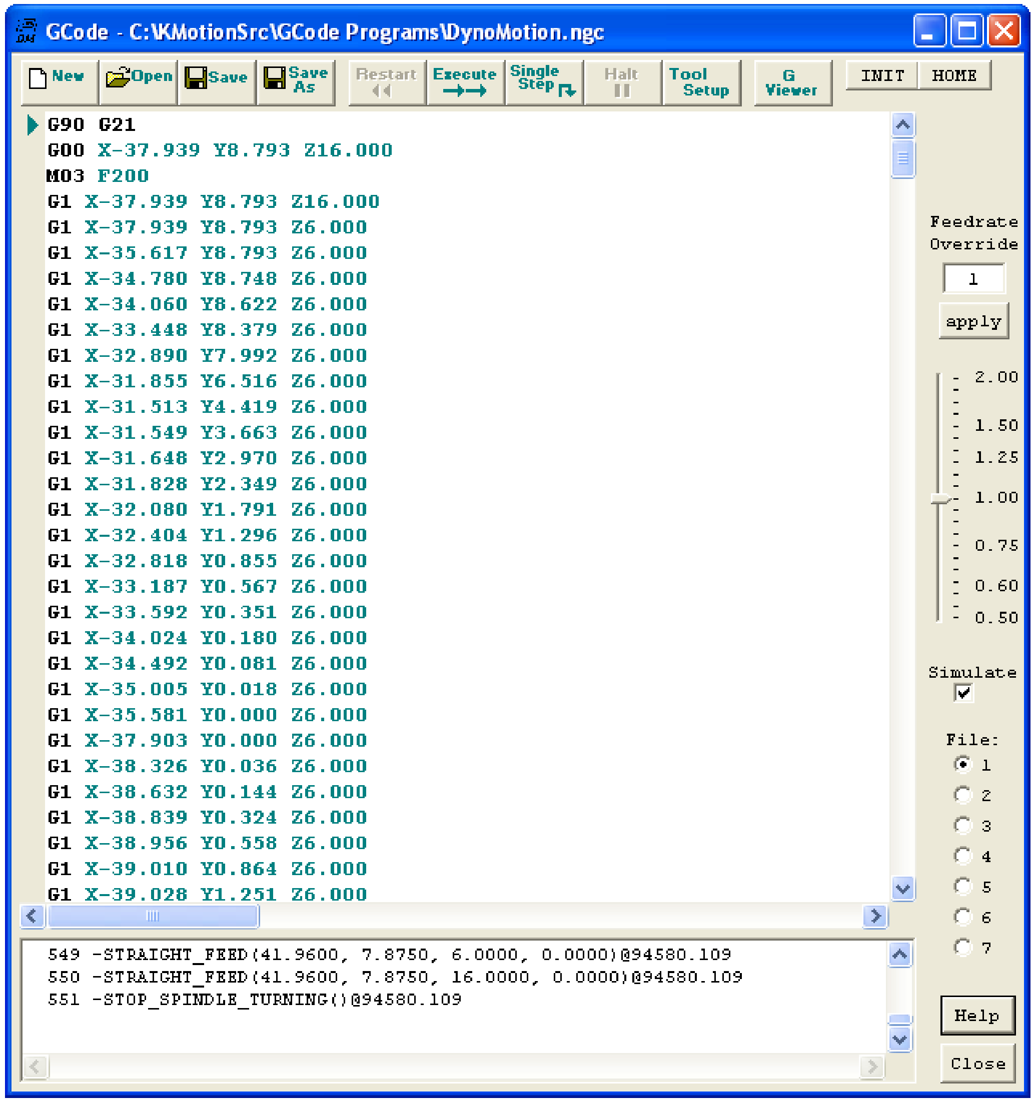
G40 Tool Comp Off
G41 Tool Comp On (Left of Contour)
G42 Tool Comp On (Right of Contour)

G43 Hn (Tool #n length comp On)
G43.4 Hn (Tool #n length comp On with RTCP On)

G49 (Tool length and RTCP 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

GCode 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.

G91 Relative Coordinates

G92/G52 Set Global Offset
G92 X0Y0 Z0
G92.1 Clear Global Offset
G92.2 Clear Leave Vars
G92.3 Load Vars

G94 Feed per min
G95 Feed per Rev

G96 Dmax CSS mode
G97 Spindle RPM mode

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 downloaded 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 xxxxxxx.

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

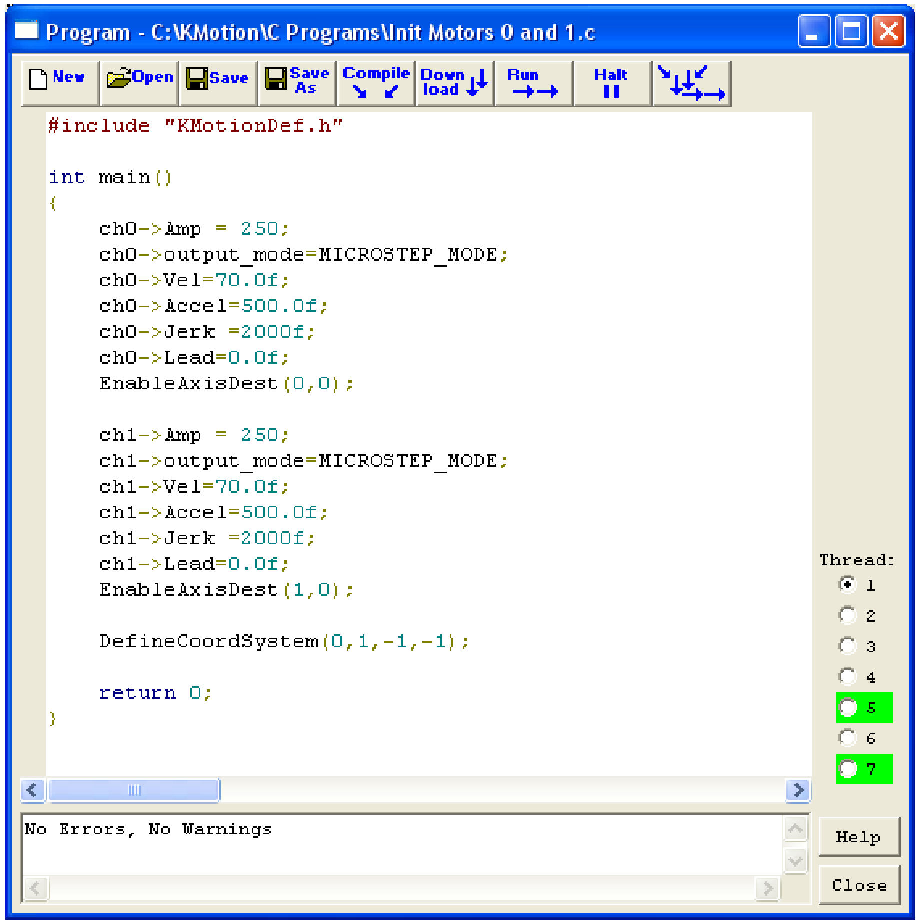
Data Gather/Plot Functions:
void SetupGatherAllOnAxis(int c,
int n_Samples);
void TriggerGather();
int CheckDoneGather();

Analog I/O Functions:
ADC(ch);
DAC(ch, value);

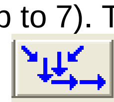
Power Amp Functions:

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:



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.

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 MoveXYZABC (double x,
double y, double z);
int CheckDoneXYZABC();
void DefineCoordSystem(int
axisx, int axisy,
int axisz, int axis a, int axis b, int
axis c);

Digitial I/O Functions:

void SetBitDirection(int bit, 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:

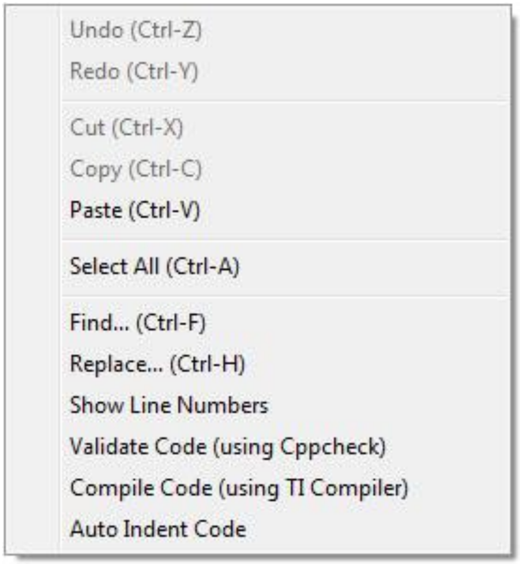
int printf(const char *format, ...);

Print to Windows File Functions:

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

Thread Functions:

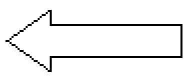
void StartThread(int thread);
void PauseThread(int thread);
void ThreadDone();
int ResumeThread(int thread);



Other features of the ***C Program Screen*** include a rich text editor with syntax highlighting, keyword drop down lists, and function tips.

Right-mouse clicking brings up the context menu (undo/redo, find/replace with regular expressions, code validator and code beautifier).

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.

TI Compiler

The Texas Instruments Optimizing Compiler is now easily usable by adding a compiler pragma at the beginning of the file. The optimization level of 0-3 can be specified as a parameter as well as the Thread Space Size if it is desirable to allow overflow of the Thread Space. Functions and Variables can also be linked to a small amount if high-speed Internal DSP RAM.

Highly optimized code can sometimes do things in parallel, in strange order, or even not at all to give unexpected results. Proper C coding with keywords such as “volatile” can inform the compiler that data can be changed by other things or has influence on things outside the current context. This ensures the compiler does things in a way that to still achieve the desired result. TCC67 does things with no optimization in the exact manner coded so avoids these issues. The TI Compiler at optimization level 0 tends to do all the things coded in the order coded also.

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);

Any detected Errors and Warnings will appear in the lower panel of the C Programs Screen. Double clicking on an error should take you directly to the line in the code where the error was detected.

The following must be the first line in the file:

```
#pragma TI_COMPILER    //defaults to optimization level 0 and normal Thread size of 0x10000 (Thread #7 = 0x50000)
#pragma TI_COMPILER(3)
#pragma TI_COMPILER(3,0x20000
```

To put a function in Fast Internal RAM:

```
#pragma CODE_SECTION(FunctionName, ".IRAM")
```

To put a variable in Fast Internal RAM:

```
#pragma DATA_SECTION(VariableName, ".IRAMD")
```

Note: \DSP_KFLOP\LinkTemplate.cmd controls default TI Linker behavior and can be changed to modify Linking behavior

KFLOP/Kogna Axis Configuration and Parameters

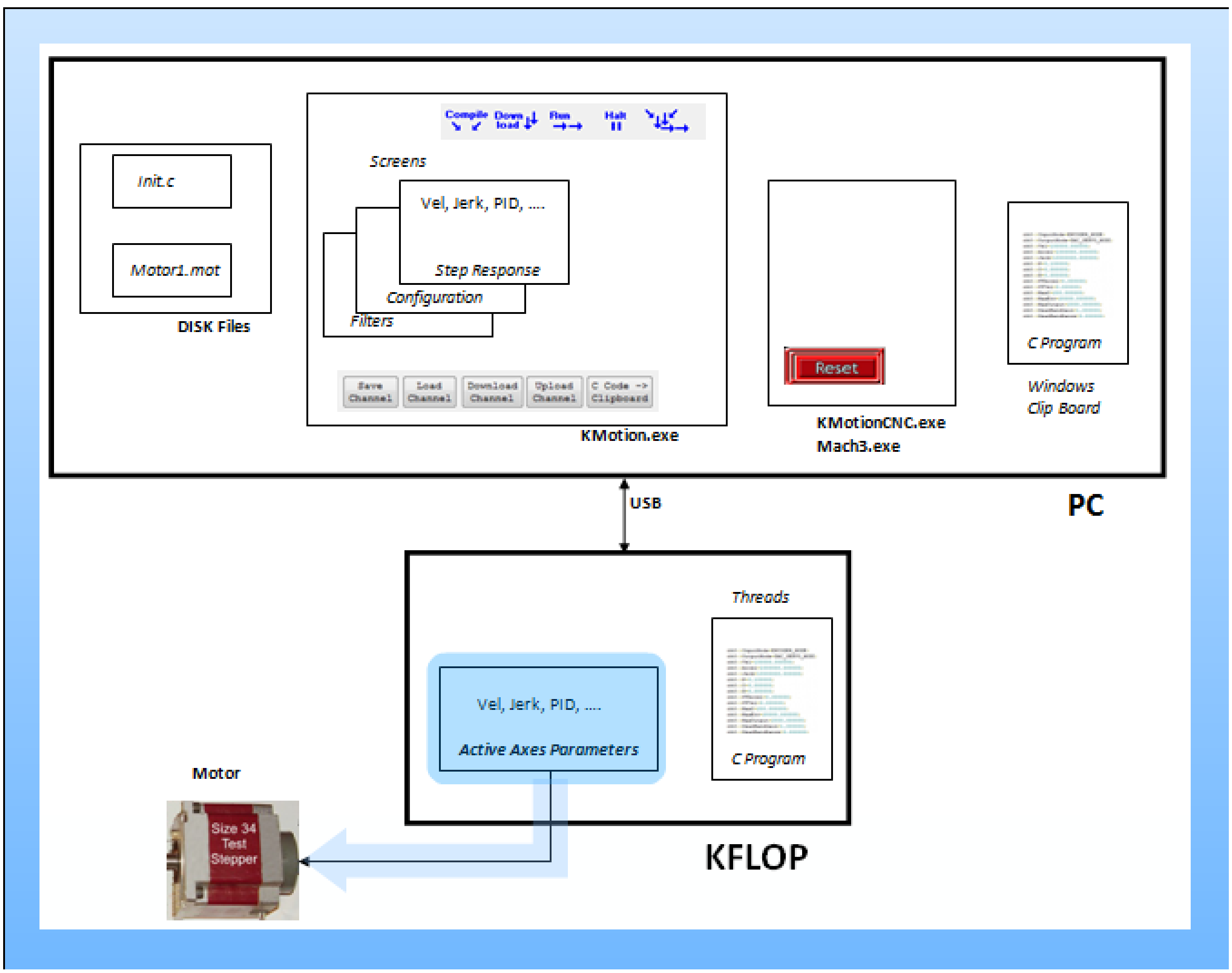
The following slides explain the relationships between the various methods of changing KFLOP/Kogna Axis Configuration and Parameter Settings.

There is one set of currently active settings within KFLOP/Kogna.

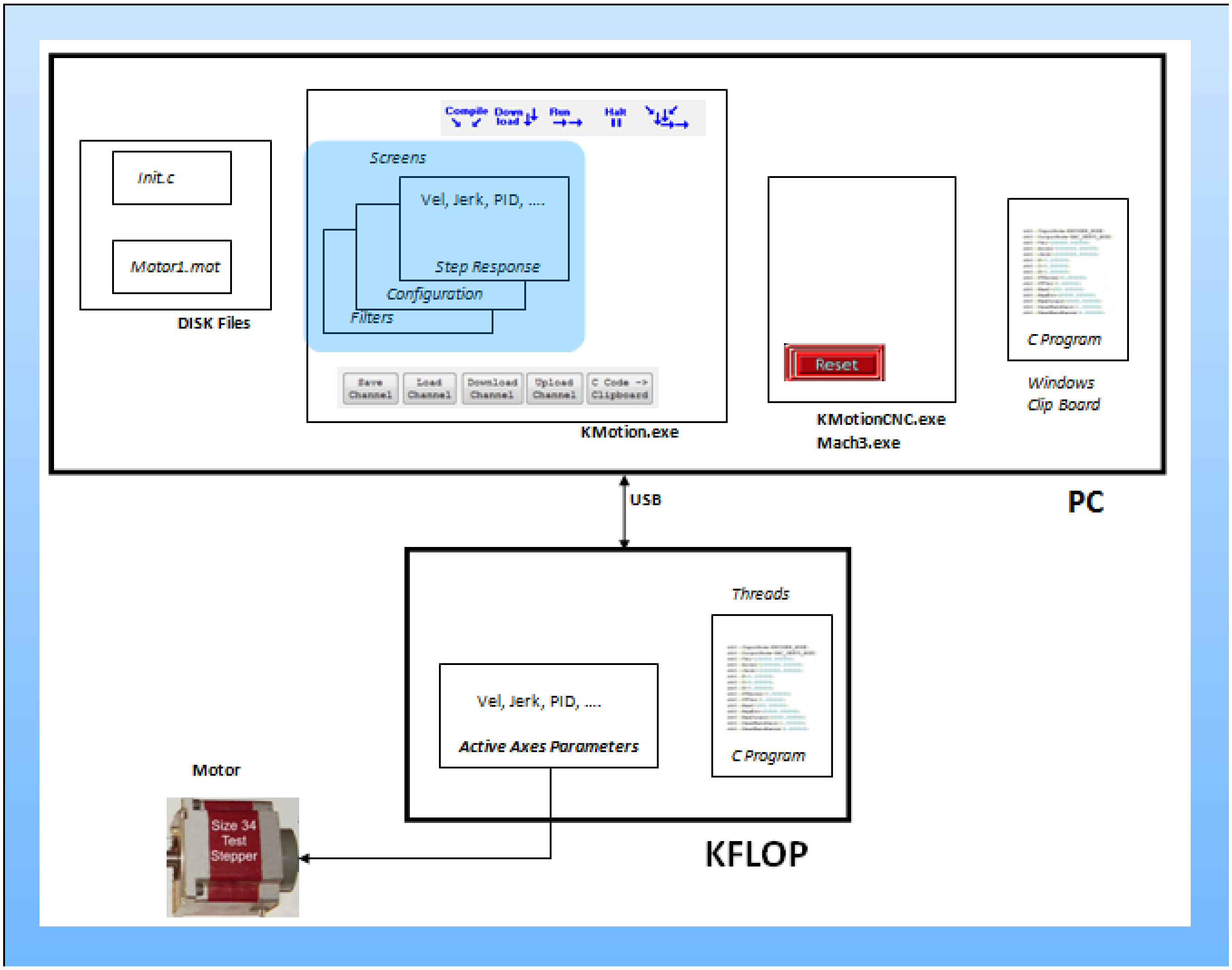
These settings can be altered in several different ways such as downloading the KMotion.exe Screems or by Executing a C Program.

The settings shown on the KMotion.exe Screens do not necessarily reflect the current settings in KFLOP/Kogna.

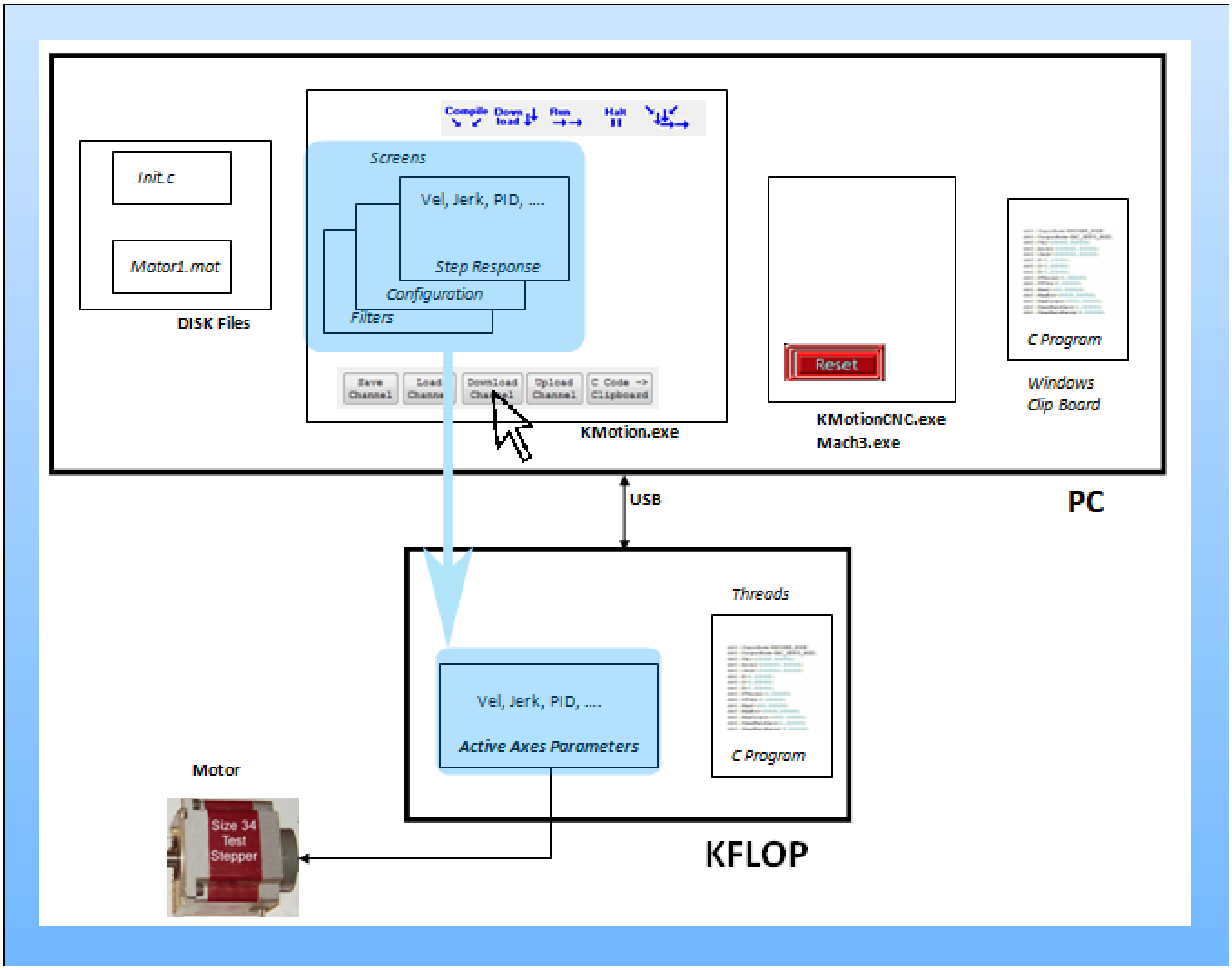
1. KFLOP/Kogna contains one set of Active Axis Parameters and Settings. These are used for any axis operations:



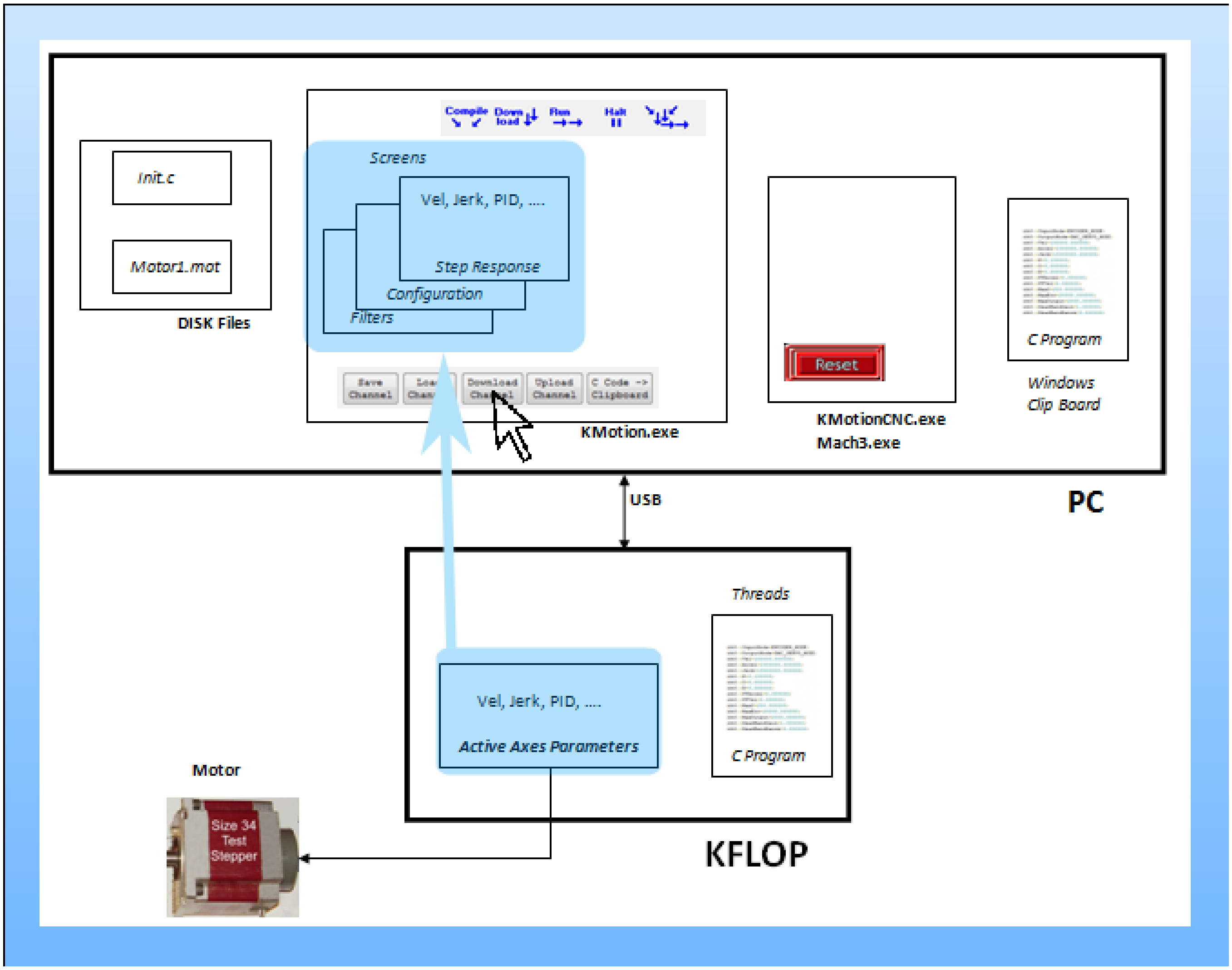
2. When the KMotion.exe Configuration and Tuning Program is running, the Configuration, Step Response, and Filter Screens have cells for all of the Axis Parameters:



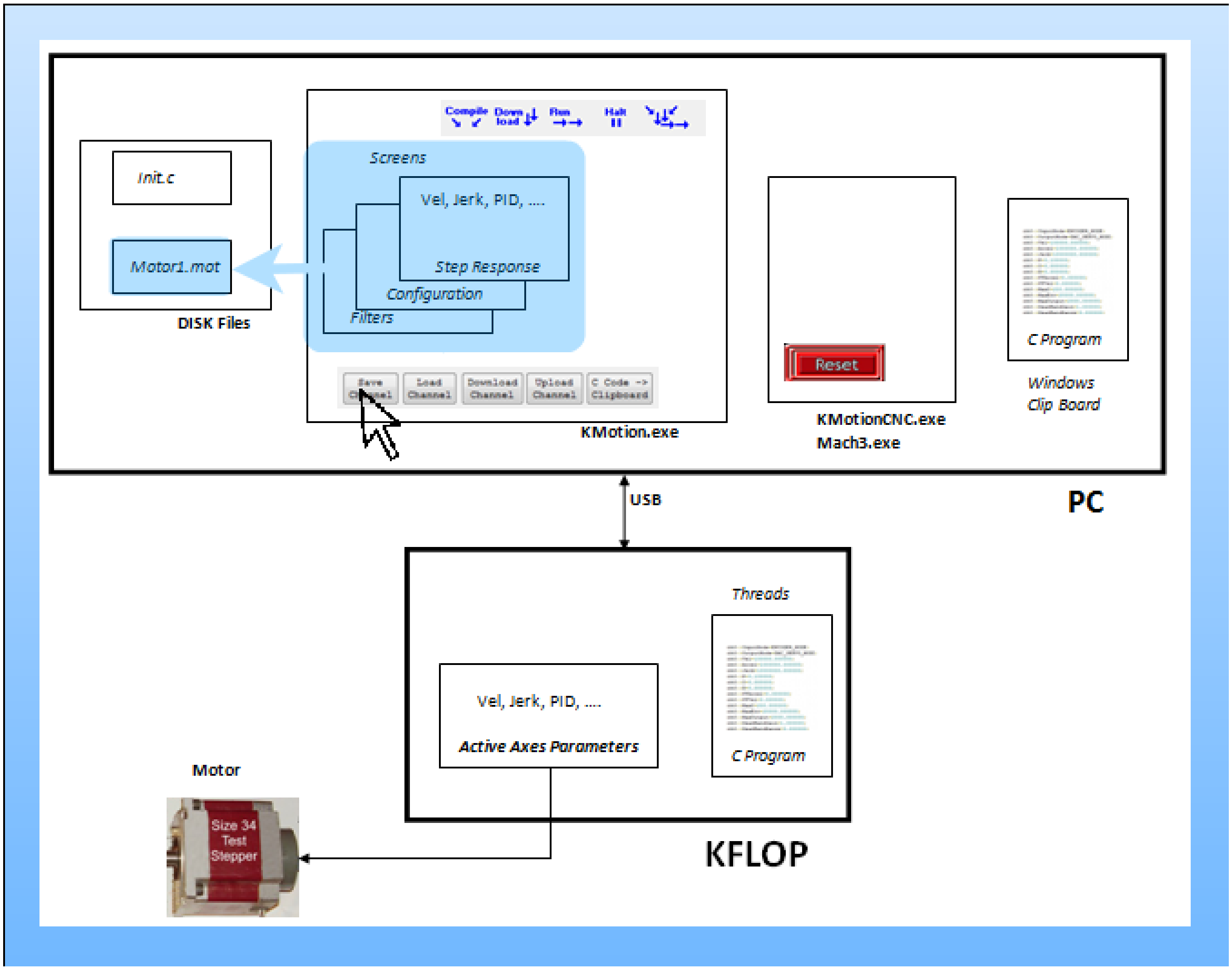
3. Pushing the "Download Channel" Button downloads the Screen settings to KFLOP/Kogna and enables the axis.:



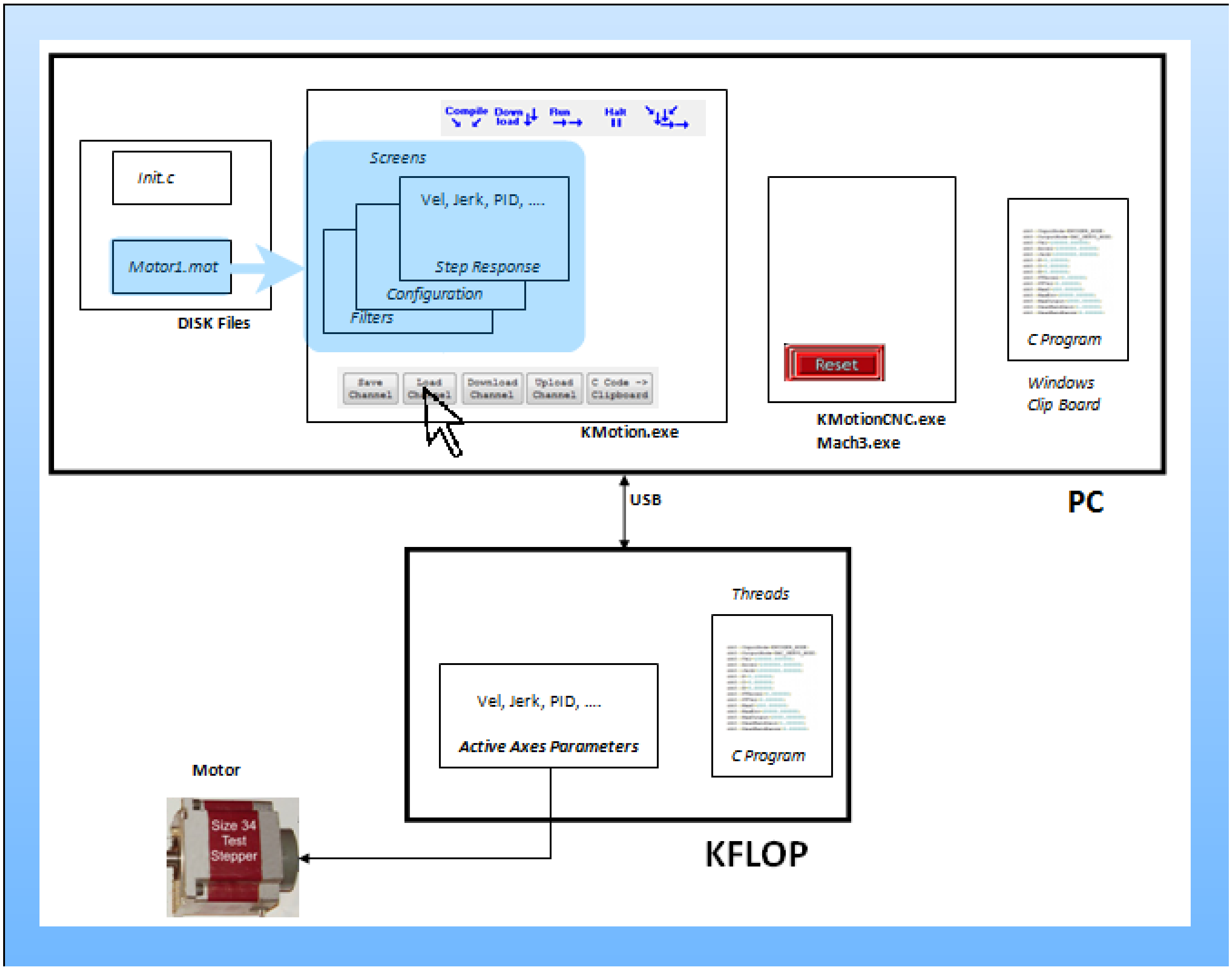
4. Pushing the "Upload Channel" Button uploads the Active Parameters from KFLOP/Kogna to the KMotion Screen Cells:



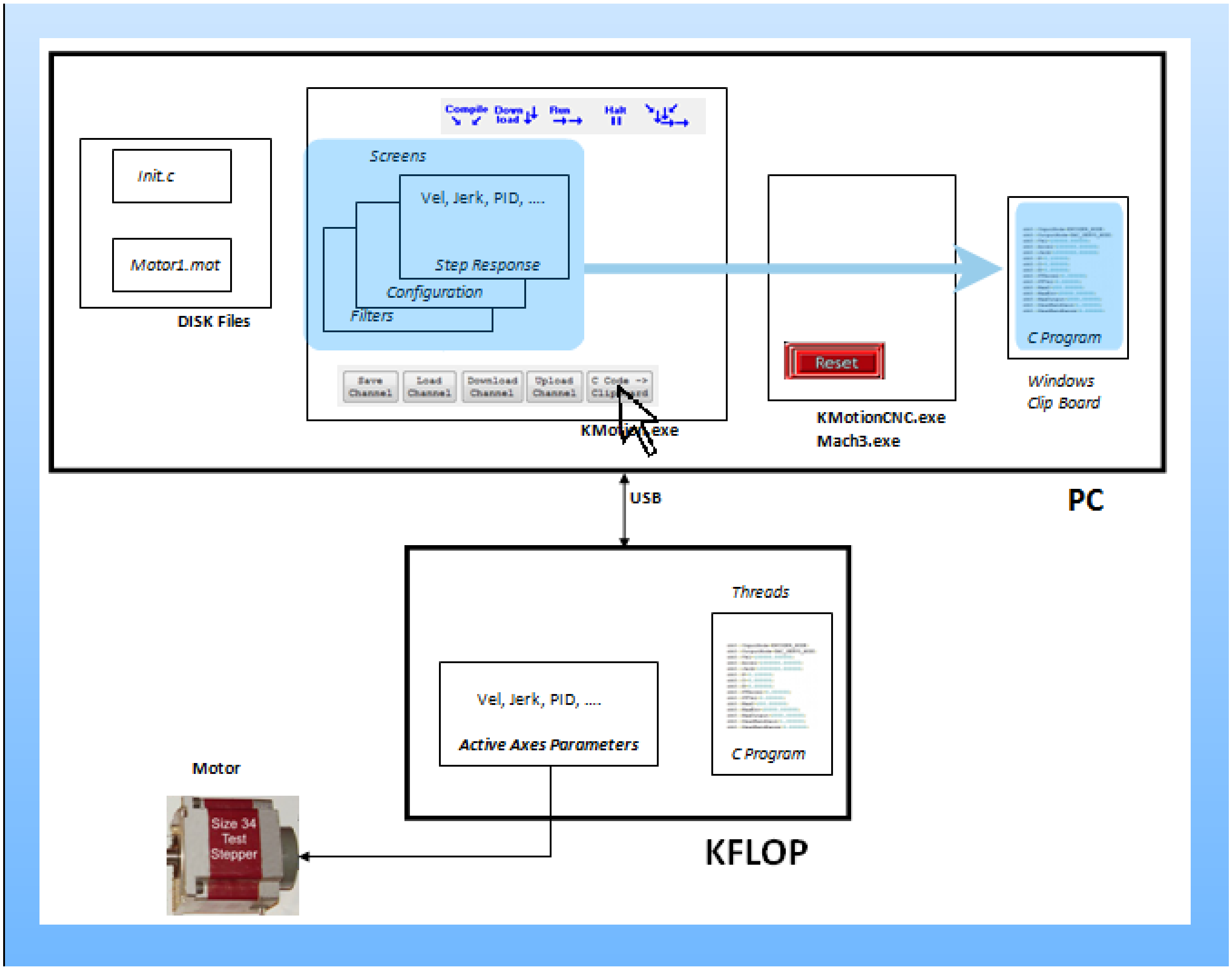
5. The KMotion.exe Screen settings for an axis can also be saved to a Disk file by pushing the "Save Channel" Button:



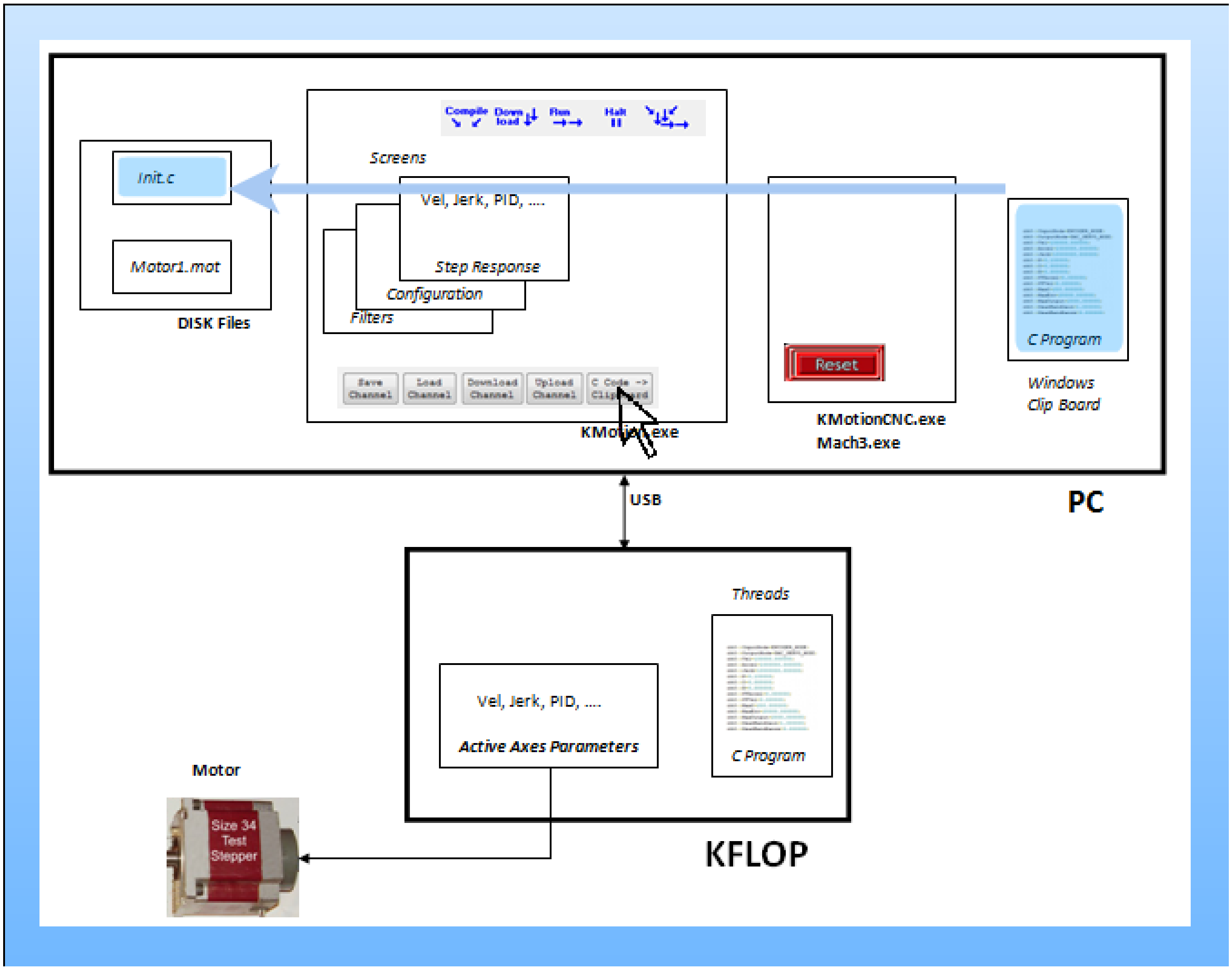
6. Or a Disk file may be loading into the KMotion Screen Cells for an axis by clicking the "Load Channel" Button:



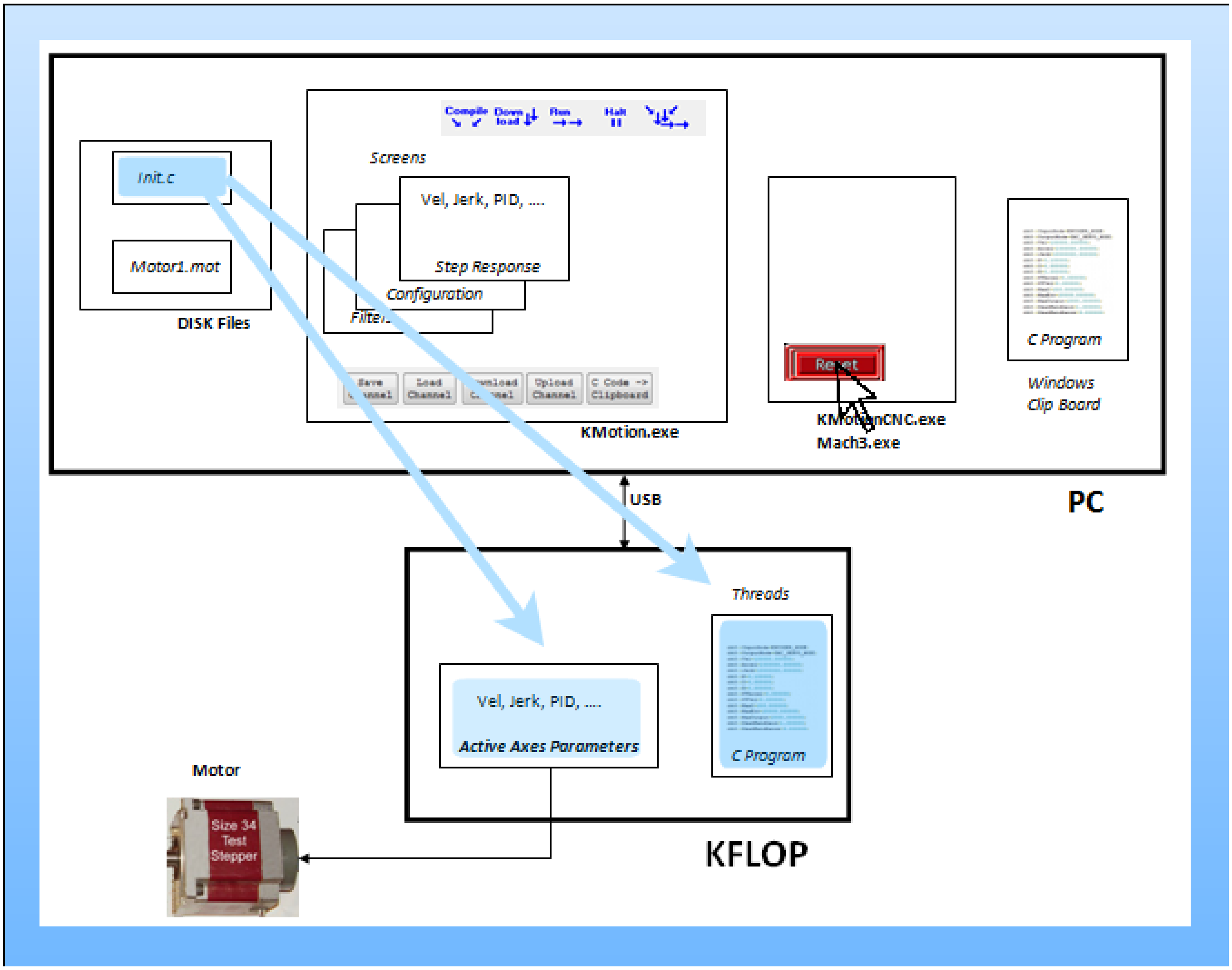
7. The Screen settings can also be converted to C Language Program Code to be later executed in KFLOP/Kogna. Push the "C Code -> Clipboard" Button to place the code into the Windows Clipboard:



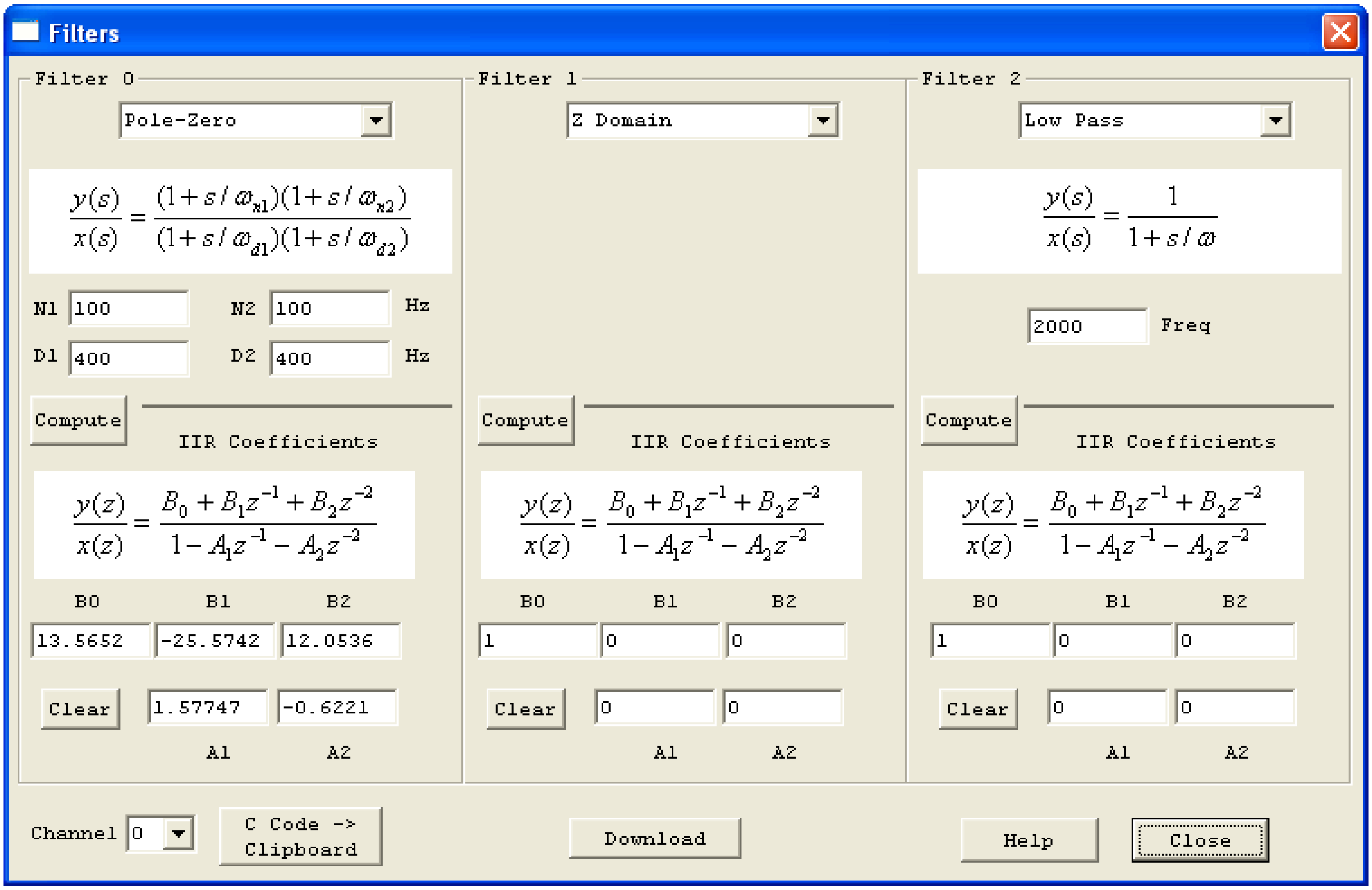
8. The C Code can then be pasted into an Initialization Program with the other axes:



9. KMotionCNC or Mach3 can then easily execute the Init.c program to set the Active Parameters in KFLOP/Kogna:



IIR Filter Screen

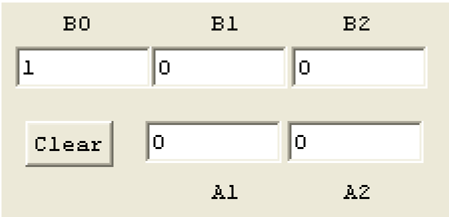


The ***IIR Filter Screen*** allows setting various IIR (Infinite Impulse Response) *Filters* into the control loop. ***KMotion*** allows up to three - 2nd order bi-quadratic stages per axis to be connected in cascade. See the KMotion Servo Flow Diagram for the placement of the IIR Filters.

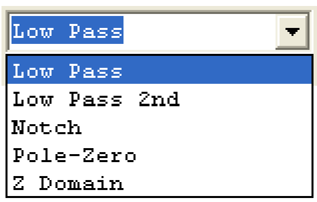
KMotion implements the filters using Z-domain coefficients shown on the bottom half of the screen. Because of the common confusion of the names and signs of the coefficients, the transfer function form is shown for reference.

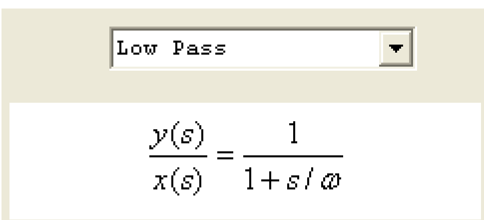
$$\frac{y(z)}{x(z)} = \frac{B_0 + B_1 z^{-1} + B_2 z^{-2}}{1 - A_1 z^{-1} - A_2 z^{-2}}$$

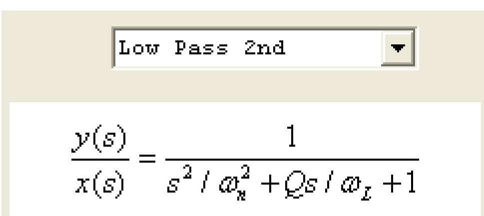
Note that setting B0=1.0 and all other coefficients, B1, B2, A1, and A2, to zero causes a transfer function of unity, effectively bypassing the filter. A Clear pushbutton is available to conveniently set this mode.

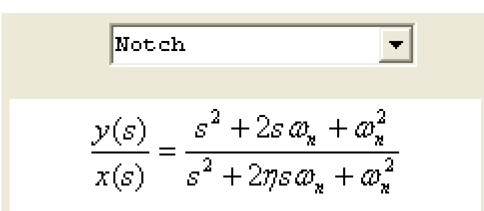


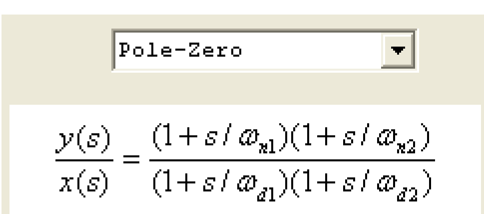
The top portion of each filter section allows various common filters to be specified in the s-domain. Supported filter types are: 1st order Low Pass, 2nd Order Low Pass, Notch, and two real poles and zeros and are selected using a drop-down list box. Z-domain is a place holder used as a reminder that 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.











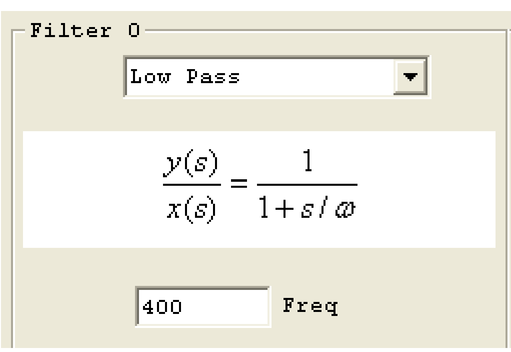
If an s-domain filter type is selected and its corresponding parameters specified, then depressing the *Compute* pushbutton will convert the s-domain transfer function to the z-domain using Tustin's approximation (with frequency pre-warping) and automatically fills in the z-domain coefficients. Note 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.

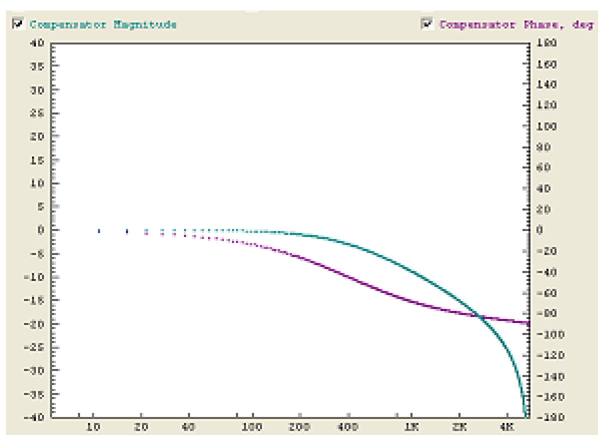
Note that the [Bode Plot Screen]() has the capability to graph the combined transfer function of all three IIR filters and the PID filter. This is referred to as the *Servo Compensation*. To view the transfer function of a *single* IIR filter, set the other filter and PID sections to unity (for PID set 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.

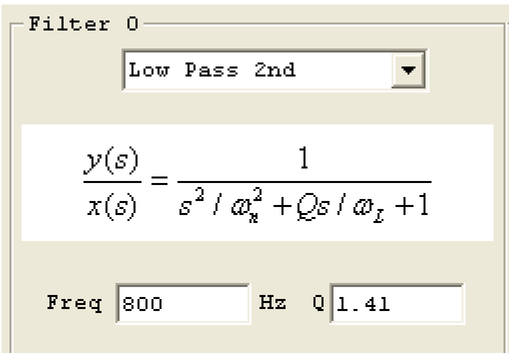


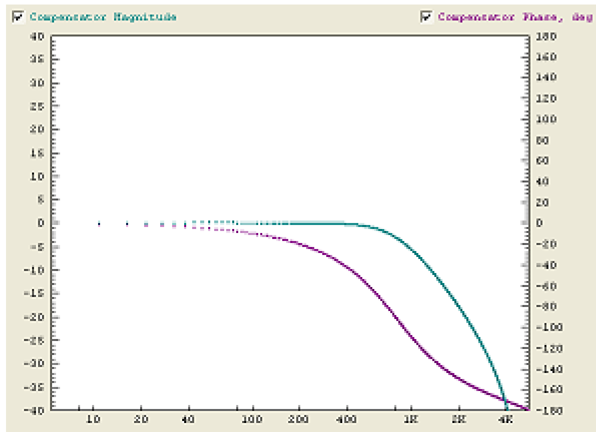


Low Pass (2nd order)

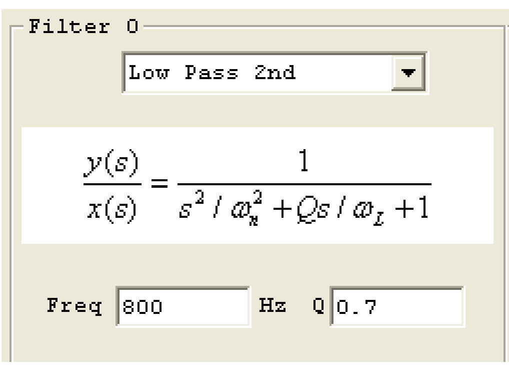
A *2nd order Low Pass* filter is commonly used in a similar manner as a [1st order low pass filter](), except that it has higher attenuation than a 1st order low pass filter. Unfortunately it also introduces more phase lag than a 1st order low pass filter. In most cases the cutoff frequency for a 2nd 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.

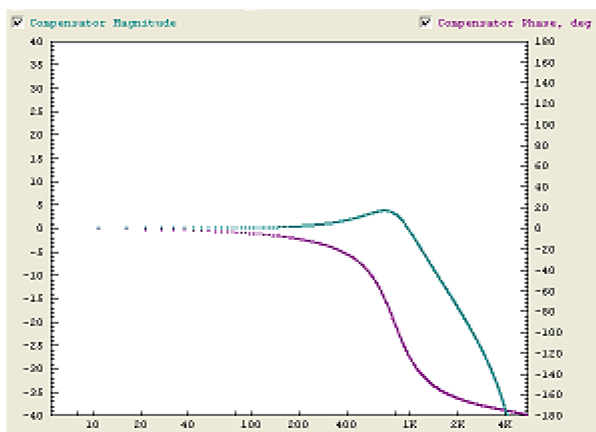
A 2nd order Low Pass filter also allows a Q parameter which changes the sharpness of the filter. A smaller value of Q results in a sharper filter at the expense of some peaking (gain increases before decreasing). A Q value of 1.41 (called a Butterworth filter), shown immediately below, is the minimal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





A Q value of 0.7 shows “peaking”.

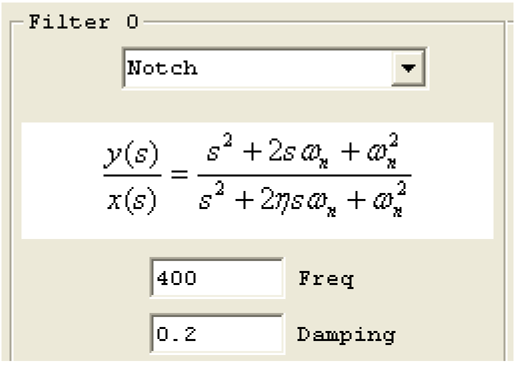


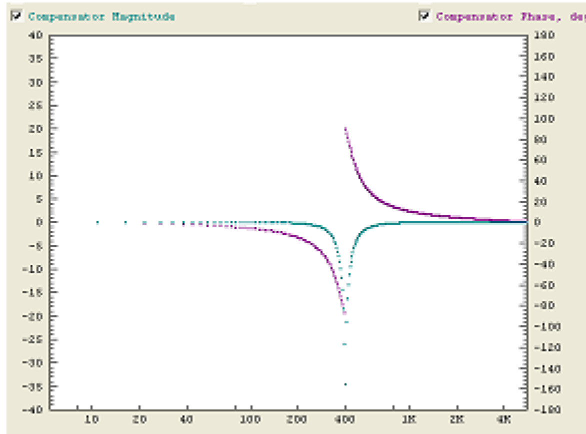


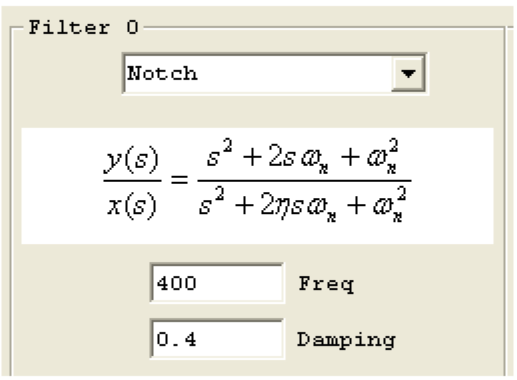
Notch

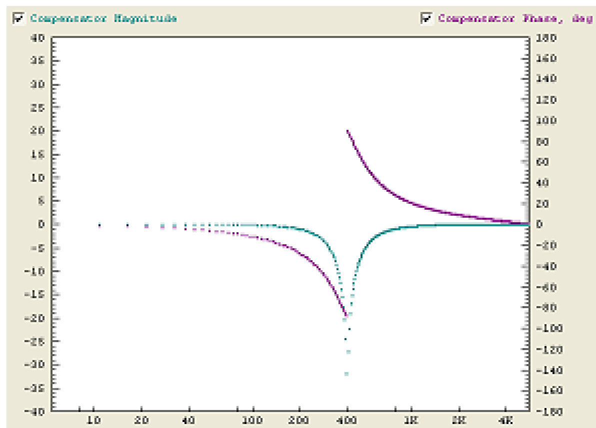
A *Notch* filter is commonly used in servo system when a sharp mechanical resonance would otherwise cause a system to become unstable. A Notch filter is able to attenuate a very localized range of frequencies. It has a damping factor, η, which effects sharpness or width of the notch. The disadvantage 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.



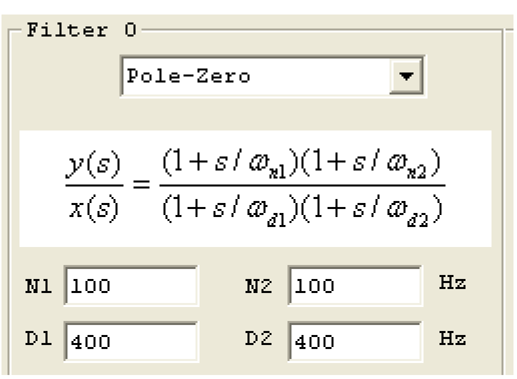


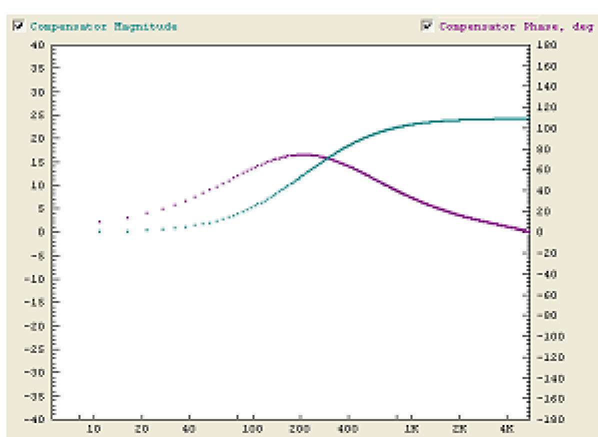




Pole-Zero

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.





Copy C Code to Clipboard

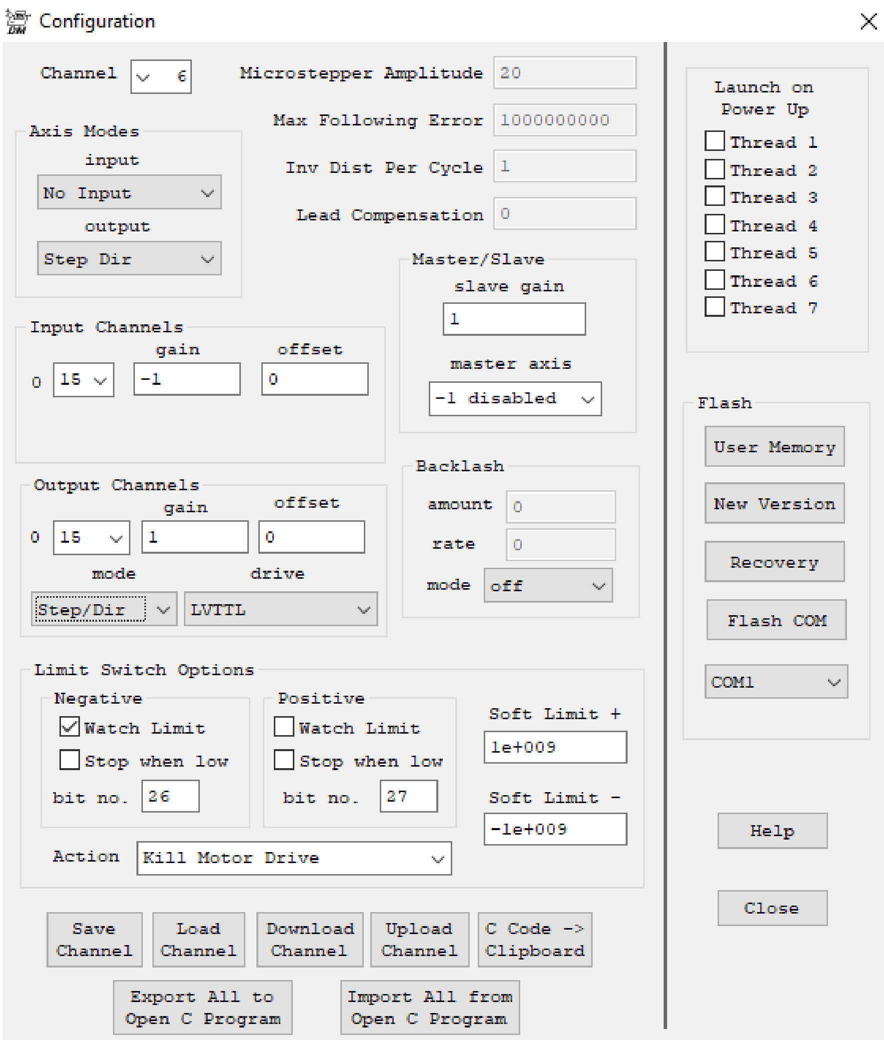
This pushbutton causes the current z-domain filter coefficients to be copied to the clipboard in a form that may be pasted directly into a ***KMotion*** C Program, see also the C Program Screen.

```
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

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



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:

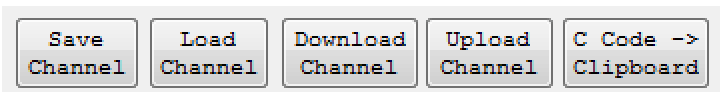
- Configuration Screen
- Step Response Screen
- IIR Filter Screen

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, intparameters are located there.

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

Utilities

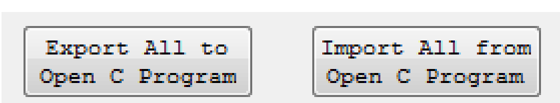


Configuration settings are normally defined and tested using the KMotion Screens. After they have been determined to work properly they can be converted to C code and placed into a C program that can then be used by applications to configure KFLOP without needing to use KMotion Screens. Axis Channel Settings can also be loaded/saved to disk files. See KFLOP Axis Configuration and Parameters .

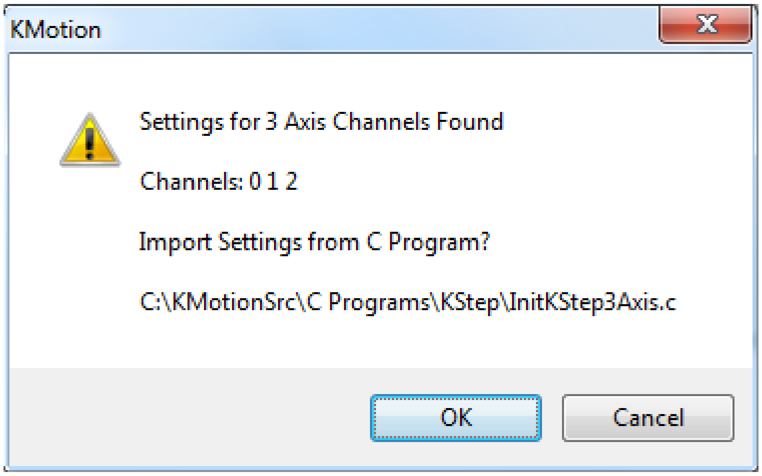
The buttons along the bottom of the Configuration Screen allow a set of axis parameters to be:

- 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 you will not see anything happen but the data will be placed in the clipboard. Paste it into a C Program to see it)*

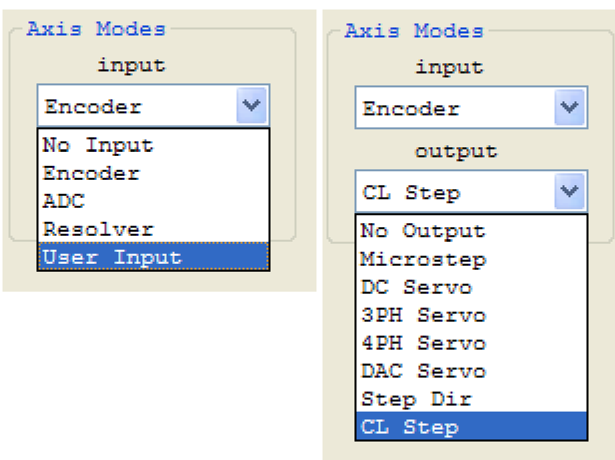
Note that these buttons operate on all parameters (for one axis) from all screens as a unit.



To completely synchronize **all** the Configuration Screens (Config/Flash, Step Response, and Filters) for **all** axes to a C Program use the **Export All to Open C Program Button**. To Import all settings from a C Program select the **Import All from Open C Program** Button. The C program must be open on the C Program Screen. In both cases the C Program will be scanned to find blocks of axis settings and will determine which axis exist in the C Program. The results of the scan will be displayed before importing or exporting the settings. The dialog box below shows an example where 3 channels were found. Select OK to proceed with the import.



Axis Modes



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, by driving a DAC signal that will drive an external power amplifier, or by driving Step and Direction digital outputs.

No Output mode is used when no output is required or a User Program will be writing the output to some device not directly supported (ie PWM Outputs).

Microstep mode is used with SnapAmp to apply sine/cosine currents to a stepper motor (this does **not** refer to driving microstepping Step Dir Drives).

DC Servo mode is used with SnapAmp to drive Brushed DC Motors

3PH Servo mode is used with SnapAmp to drive Brushless 3 Phase Motors.

4PH Servo mode is used with SnapAmp to drive Brushless 4 Phase Motors (ie Steppers as servos)

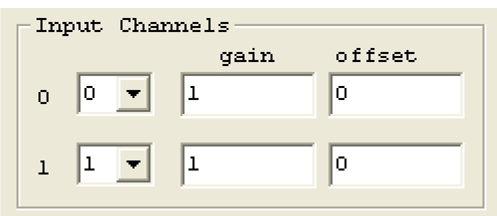
DAC Servo mode is used with Kanalog DACs to drive Analog Amplifiers.

Step Dir mode is used with open loop Step/Direction Drives (ie KStep or other external Drives). Note KFLOP outputs "steps" to external drives and whether the drive interprets the "steps" as microsteps or full steps or servo steps are dependent on the Drive.

CL Step mode is used with Step/Direction Drives in a closed loop manner with feedback.

For External Step and Direction Outputs see [Step and Direction Output Mode]() and [Closed Loop Step/Dir Output Mode]().

Input Channels



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 my 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 require two ADC input channels (for sine and cosine), for all other modes the second channel number is not used.

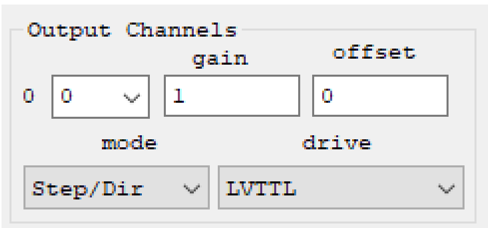
The gain and offset parameters are applied to the respective input device. The gain is applied before the offset, i.e. $x' = ax+b$, where a is the gain and b is the offset.

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 it's 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 ***KFLOP*** 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. ***KFLOP*** 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 and desired range.

Output Channels



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 my 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.

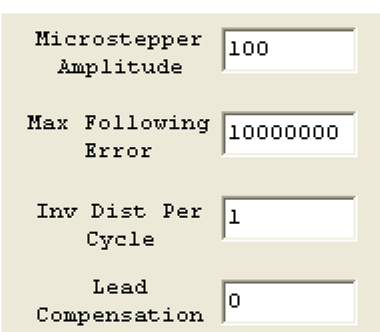
DC Servo motor (Brush motor type) only require one PWM channel.

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 is only applicable to configurations with output mode of Microstepper with SnapAmps (Not Step/Dir Drives). 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 ***KFLOP*** 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, DAC Servo, and Closed Loop Step/Dir). 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.

Max Following Error can be used to protect the system if anything goes wrong. For example if hitting an obstacle, servo going unstable, drive disabled, loss of motor power, etc. the Position will not be able to follow the trajectory and a relatively large error will occur and the axis will disable. This avoids any violent corrections that might otherwise occur. The Step Response Screen might be used to determine the maximum following errors for the axis under all normal conditions. Then set *Max Following Error* to a somewhat larger value. For example if the axis normally always follows with less than 100 counts of error the *Max Following Error* might then be set to 150.

If Following Errors occur under normal operations, the axis should be either tuned better so that following errors are reduced, or Velocity, Accelerations, and/or Jerk settings reduced to make the trajectories less demanding to follow. Otherwise the *Max Following Error* should be increased if such errors are tolerable for the application.

Inv Dist Per Cycle applies to Stepper, 3 Phase, and 4 Phase motors driven by SnapAmps. 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 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.

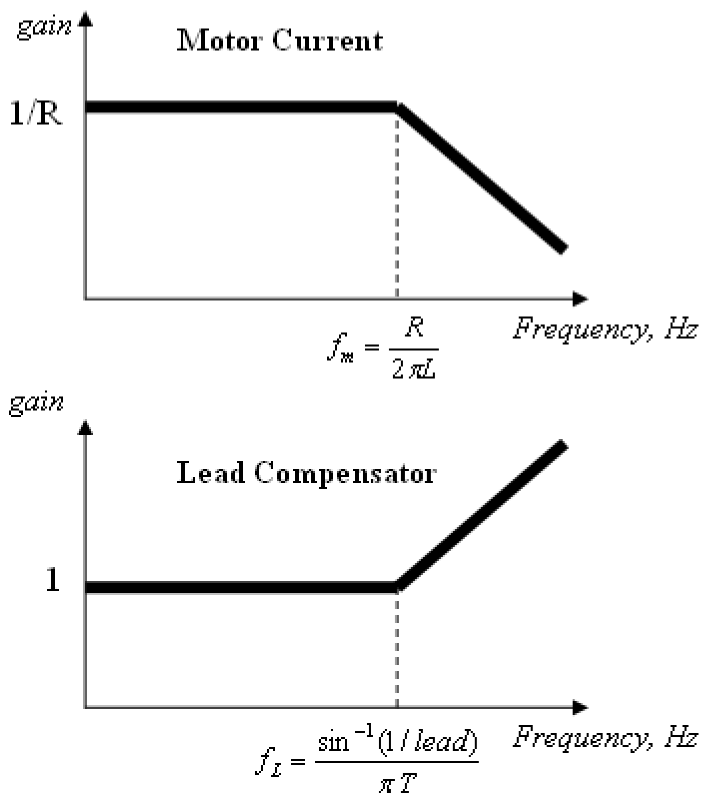
For 3 Phase or 4 Phase motors, *Inv Dist Per Cycle* represents the inverse of the distance for one complete commutation cycle. See the example below.

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). ***KFLOP*** 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 by the coil's resistance and is constant. As the frequency increases at some point, where $f_m = \frac{R}{2\pi L}$, the inductance, *L*, begins to dominate and the current drops (see plot below). ***KFLOP'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 + Δv 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).

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

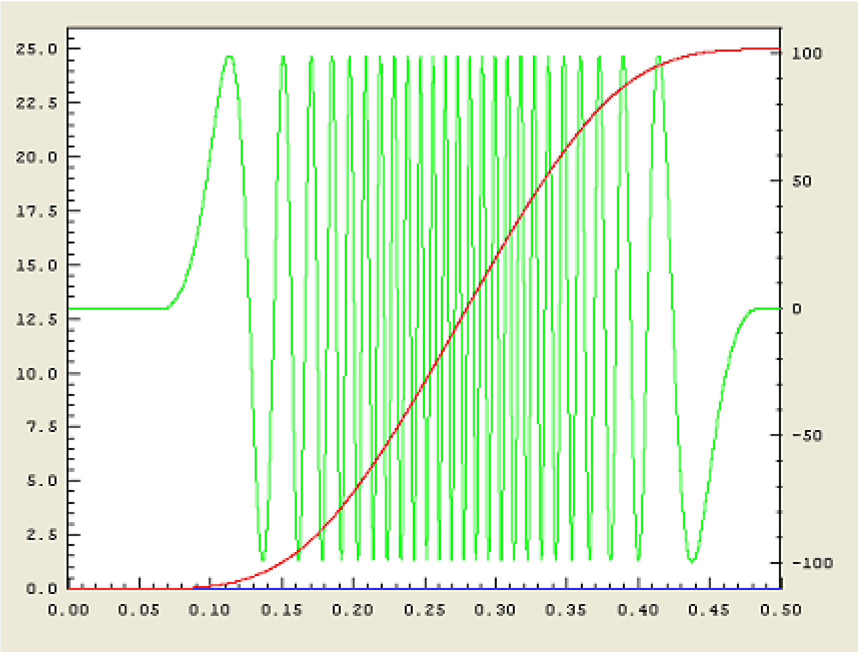
or the inverse of this formula will provide the lead value to position the knee at a particular frequency.

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

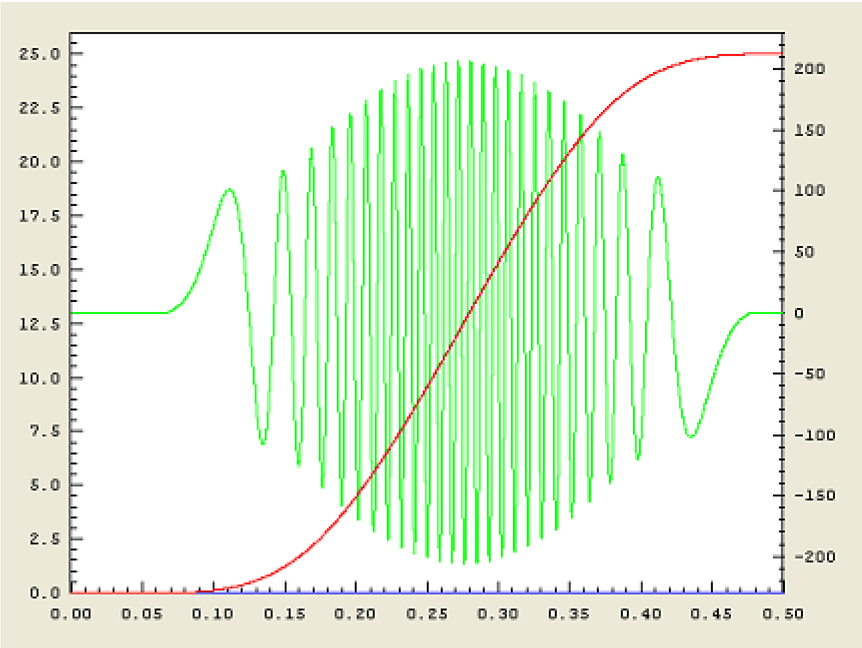
The Following table generated from the above formula may also be used. For most motors the Lead Compensation values will be within the range of 5 - 20.

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/sec^2, Jerk=10000 cycles/sec^3. Note that regardless of velocity PWM amplitude is constant.



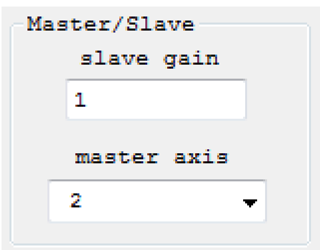
This plot displays the same 0.5 second motion with Lead Compensation = 27.0. All other parameters same as above. Note how PWM amplitude increases with velocity

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

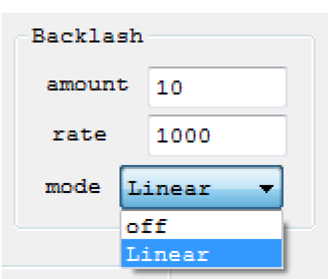
Setting the Lead Compensation too high should be avoided, as it may cause over current in the motor at medium speeds or voltage distortion due to saturation (clipping).

Master/Slave Settings



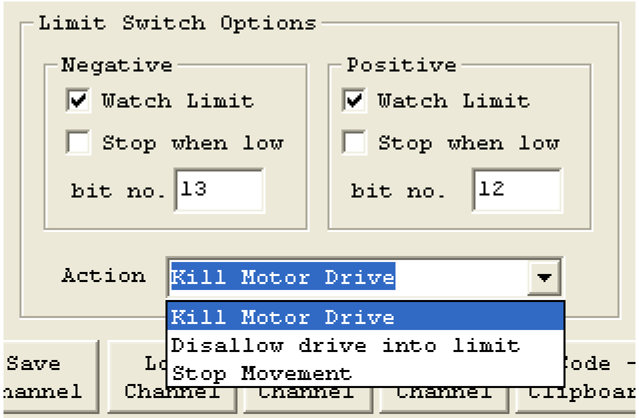
Configures the axis to be slaved to another axis. If slaved when the master axis moves, this axis will be commanded to move by an amount as scaled by the slave gain. If the Slave Gain is negative the slaved axis will move in the opposite direction as the Master. See also Console commands SlaveGain and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

Backlash Settings



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



KFLOP 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.

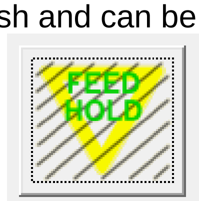
The Action drop down specifies what action should be performed when a limit is encountered.

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 FeedHold the Coordinate System. This will cause commanded positions to decelerate to a stop in a controlled manner. Independent motions will decelerate to a stop in the same manner as a Jog to zero speed would cause. Coordinated Motion will decelerate all axes to a stop along the motion path.

The FeedHold mode will remain and prevent any further motion until cleared. In KMotion.exe the 

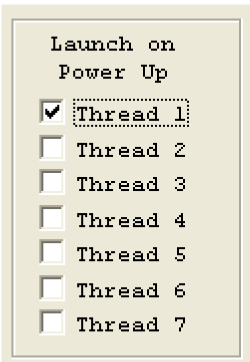
button will flash and can be pushed to clear the Feedhold. In KMotionCNC Feedhold can be cleared by pushing 

. If further motion is attempted into the Soft Limit another Feedhold will occur. However, if no motion, or motion out of the Limit, Feedhold will remain clear and the motion will be allowed.

Soft Limits

Soft Limits will always prevent motion in the same manner as a Hardware Limit with the Stop Movement Action Selected. This occurs regardless of the Action Type Selected for the Hardware Limit Switches. To disable Soft Limits set them to a huge range which could never occur. Soft Limits prevent motion within KFLOP when Jogging, moving and so forth. They also are are uploaded by Applications such as KMotionCNC and used to prevent motion during Trajectory Planning.

Launch on Power Up

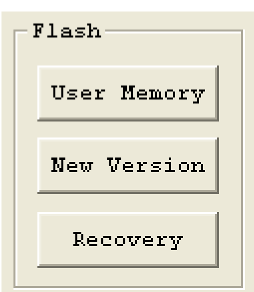


The launch on power up configuration specifies which User Programs are to be automatically launched on power up for stand alone operation of ***KFLOP***. See the C Program Screen for information on how to Edit, Compile, and Download a C program into ***KFLOP*** for execution into one (or more) of the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

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 ***KFLOP***

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, ***KFLOP*** 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).

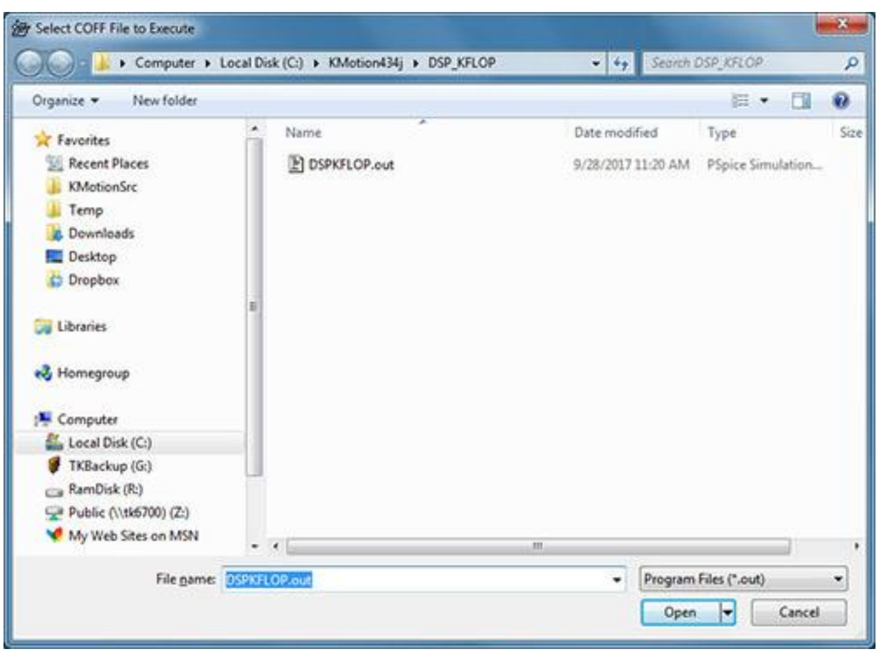
To upgrade the system firmware in a ***KFLOP*** use the *Flash - New Version* button. The user will be prompted to select a **DSPKFLOP.out** COFF file from within the ***KMotion*** Install Directory to download and Flash. Note that all user programs and data will be deleted from ***KFLOP*** when loading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

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

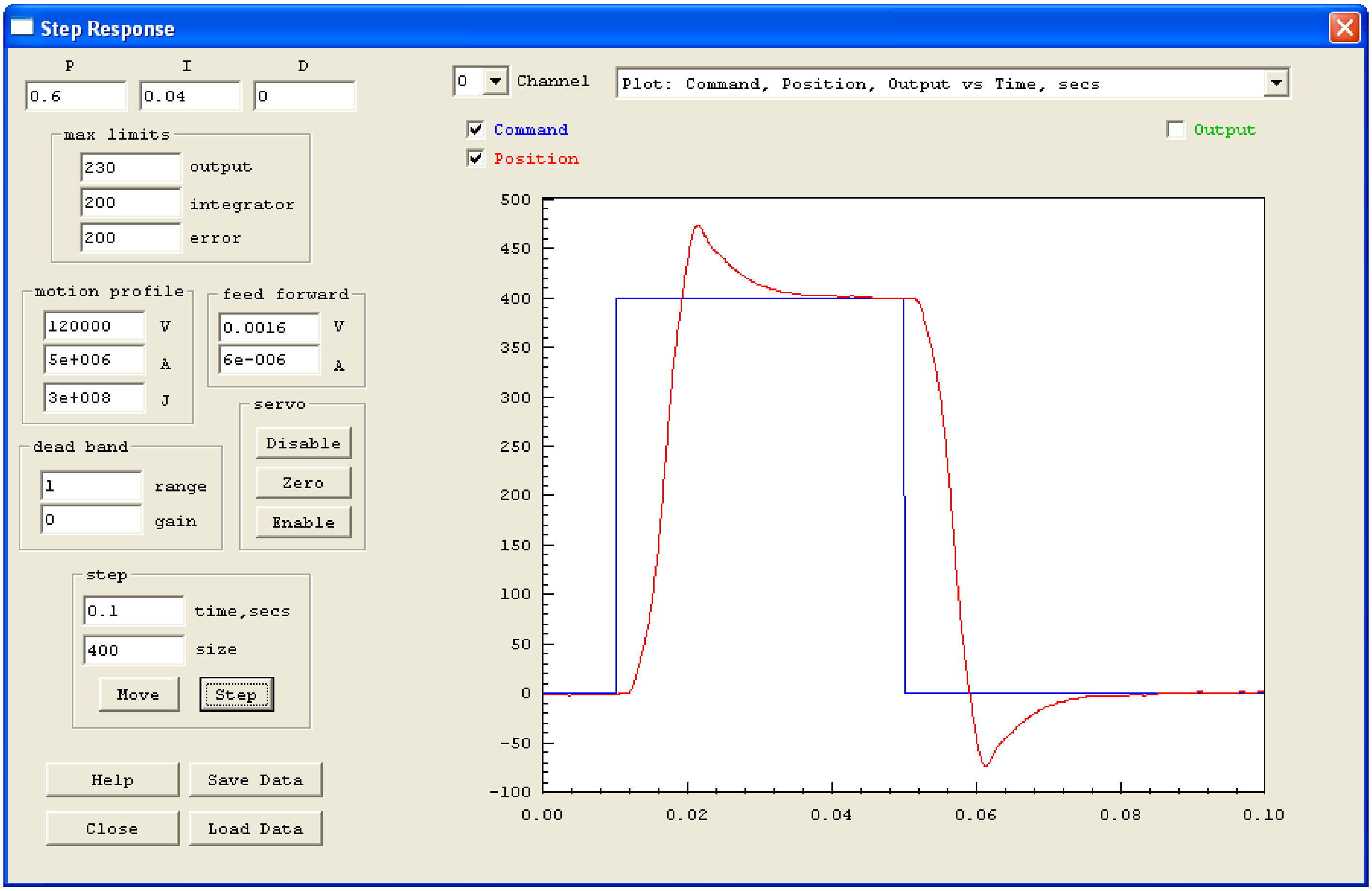
It is important that the **<Install Directory>\DSP_KFLOP\DSPKFLOP.out** file match the firmware that is flashed into ***KFLOP***. User C programs are *Linked* using this file to make calls and to access data located within the ***KFLOP*** firmware. Whenever a user program is compiled and linked using this file, the timestamp of this file is compared against the timestamp of the executing firmware (if a ***KFLOP*** 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:

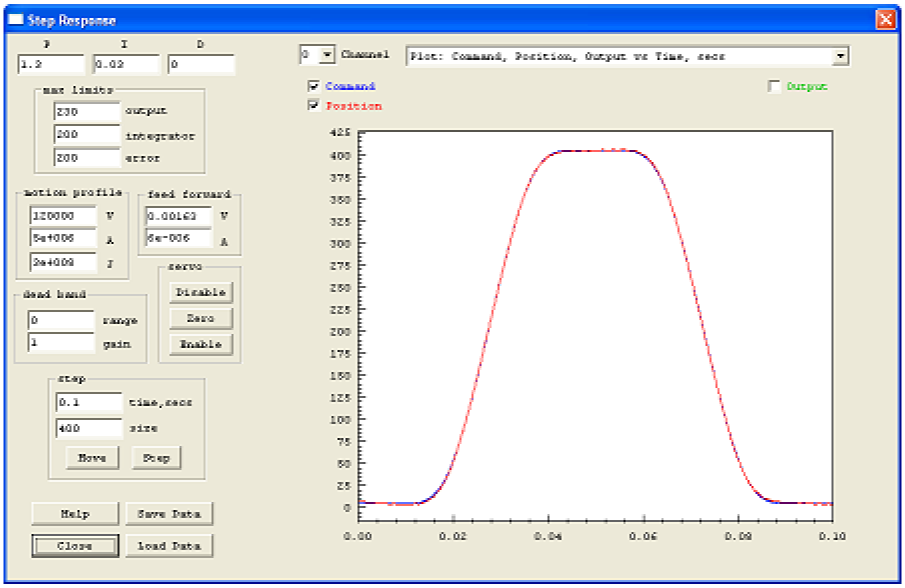
1. Turn off ***KFLOP***
2. Run ***KMotion.exe*****,** Config/Flash Screen, select "Recovery"
3. Click OK (see below)
4. Select the **DSPKFLOP.out** file to boot (see below)
5. Click OK
6. When prompted, turn on ***KFLOP***
7. After ***KFLOP*** boots, Flash the New Version

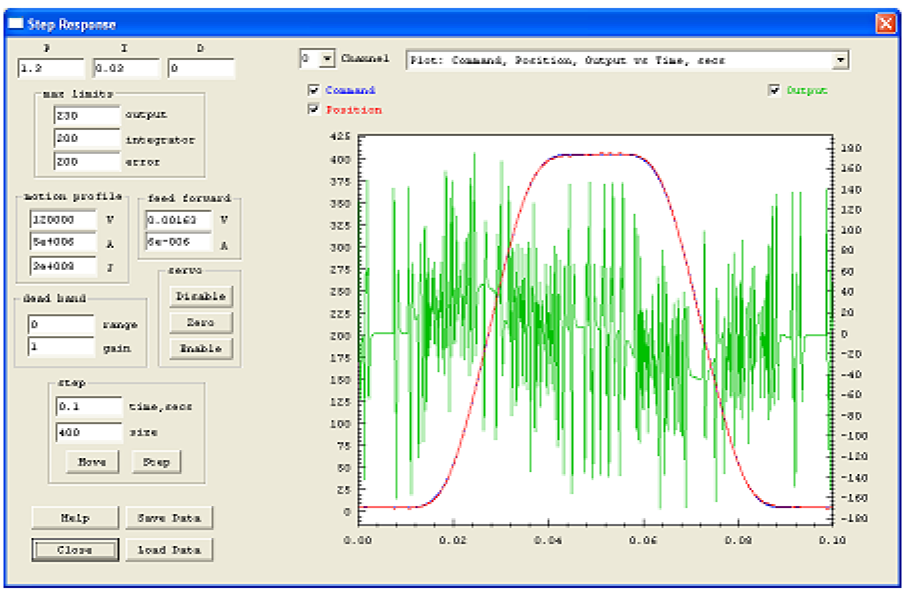


Step Response Screen

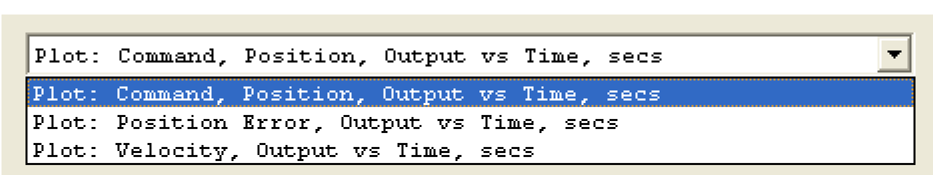


The ***Step Response Screen*** allows changes to system tuning parameters and allows measurement and graphs of the system's time response for either a Move Profile or a Step Function. The graph shown above is of an applied step function of 400 counts. The graphs shown below are of a profiled move (and 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.



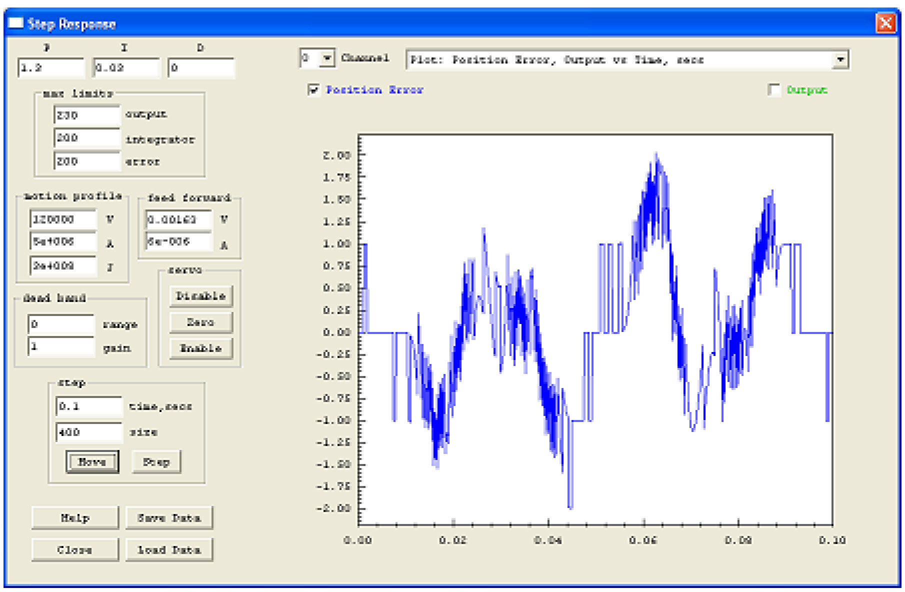
They consist of either:

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

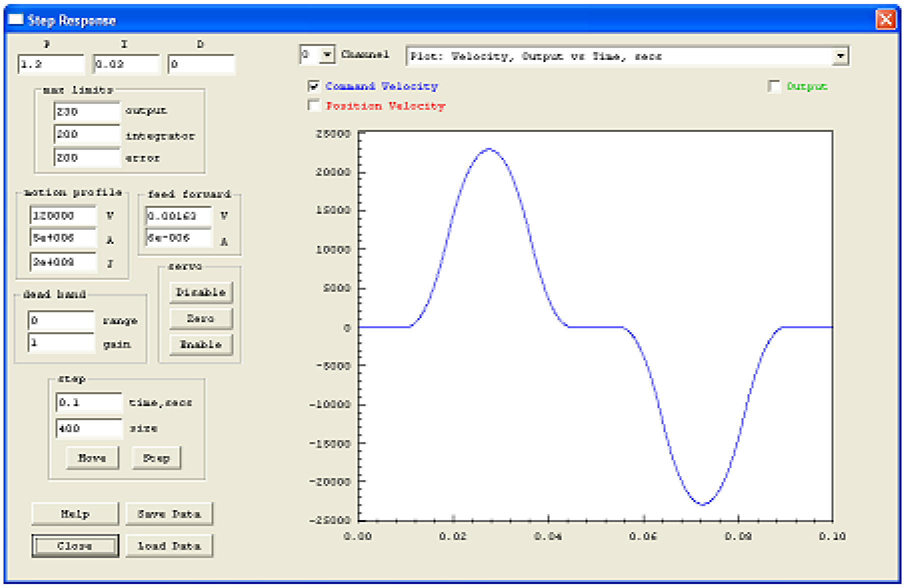
For all three plot types the Motor Output is always displayed as a secondary *Y* axis on the right side of the graph. The other plotted values are on the primary left *Y* axis. The *X* axis is always time in seconds. After a particular plot type has been selected, each individual plot variable may be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

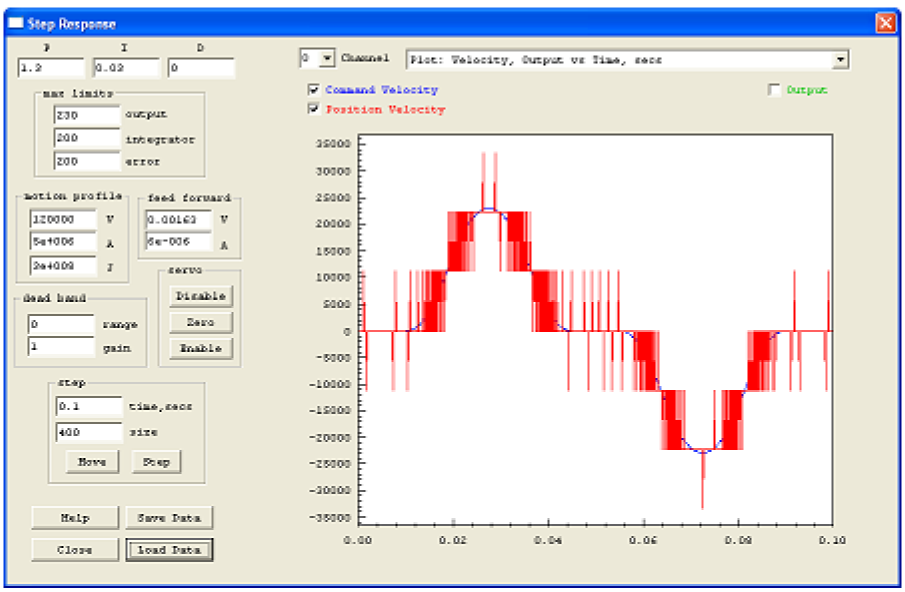
Any portion of the graph may be zoomed by left-click dragging across the graph. Simply select the area of interest. Right clicking on the graph will bring up a context menu that allows zooming out completely or to the previous zoom level.

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.

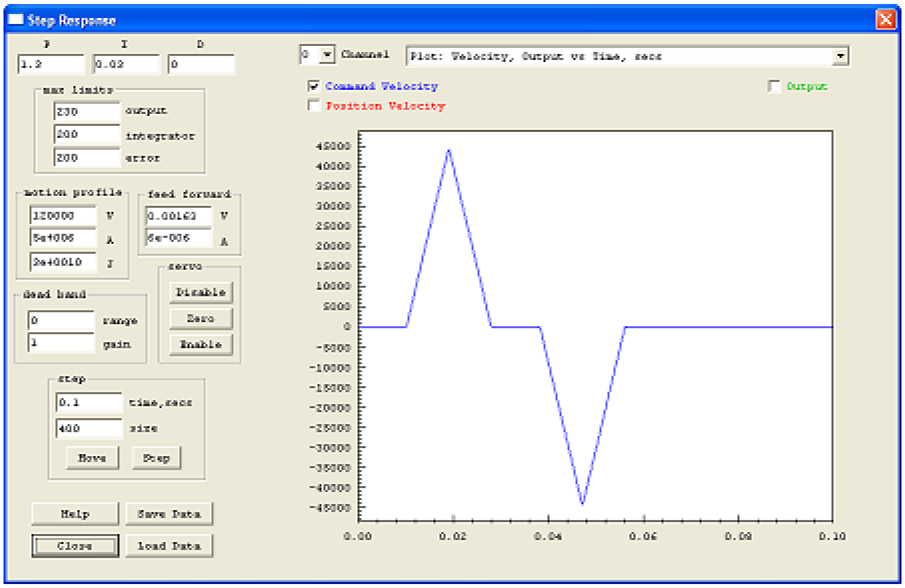


The third type of plot displays the Velocity of the Commanded and/or Measured Position. Velocity units are *Position Units* per second. When a Move is commanded, a motion profile is computed which achieves the motion in the shortest time without exceeding the maximum allowed velocity, acceleration, or 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.

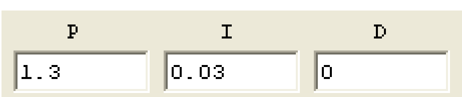




The velocity graph below, shows the effect of setting the maximum allowed jerk to a very large value (100X higher than the graph above). Note how the slope of the velocity changes abruptly which represents a high rate of change of acceleration (jerk).



Tuning Parameters – PID



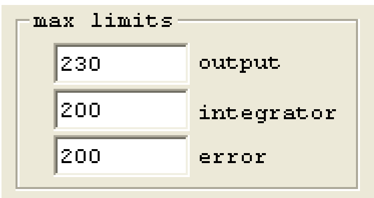
The PID (proportional, integral, and derivative) gains set the amount of feedback of the error itself (proportional), the integration of the error (integral), and the derivative of the position (derivative) that is applied to the output. Also see the [KMotion Servo Flow Diagram].

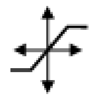
The units of the proportional gain are in Output Units/Position Units. For example if the Position Units are in encoder counts, and the Output Units are in [PWM counts], then a gain of 10.0 would apply an output drive of 10 PWM for an error of 1 encoder count.

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 an 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.

Tuning Parameters - max limits



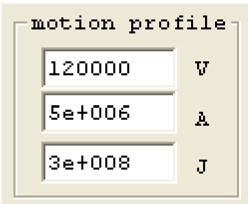
KMotion's *max limits* allow several points in the Servo Flow Diagram to be clamped, or limited to a specified range. The limits in the flow diagram are shown as a clamp symbol  . This capability is often useful in controlling how the system responds to extreme situations.

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.

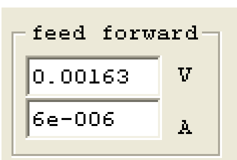
Maximum *error* limits the maximum value allowed to pass through the servo compensator. The units are the same as position units. Typically, when a servo loop is operating normally, its following error is a small value. When some extreme even occurs, such as a sudden large step command, or possibly a 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



The Motion Profile parameters set the maximum allowed velocity (in position units per second), the maximum allowed acceleration (in position units per second^2), and the maximum allowed jerk (in position units per second^3). 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.

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.

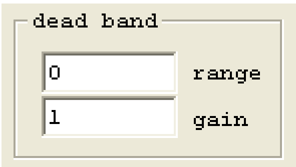
Most motion systems are constructed in manner where some sort of motor force is used to accelerate a mass. In these cases whenever an acceleration is required a force proportional to the acceleration will be required to achieve it. Acceleration feed forward may be used to reduce the amount that the feedback 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.

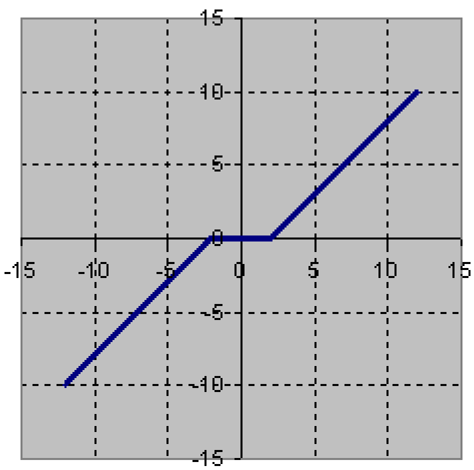
The normal procedure to optimize feed forward is to select plot type - position error, and measure moves using the *Move Command* (Step functions should *not* be used as step functions are instantaneous changes in position that represent infinite velocity and acceleration).

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

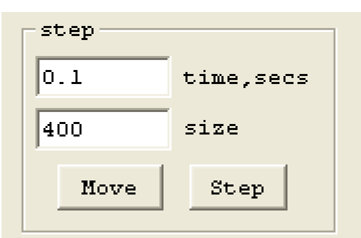


Dead band is used to apply a different gain to the region near zero than the rest of the region. Usually either zero gain or a gain much less than 1 is used within the dead band range. See the [Servo Flow Diagram] for the exact location of where the dead band is inserted. Dead band is a means of introducing "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



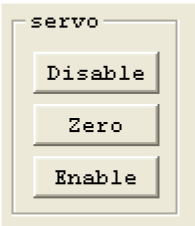
To perform a measurement and display the response, select the time duration to gather data, and the move or step size to perform, and press either the Move or Step buttons. If the axis is currently enabled, it will be disabled, all parameters from all screens will be downloaded, the axis will be enabled,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

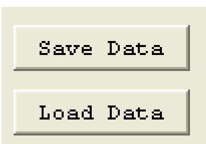
Axis Control



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

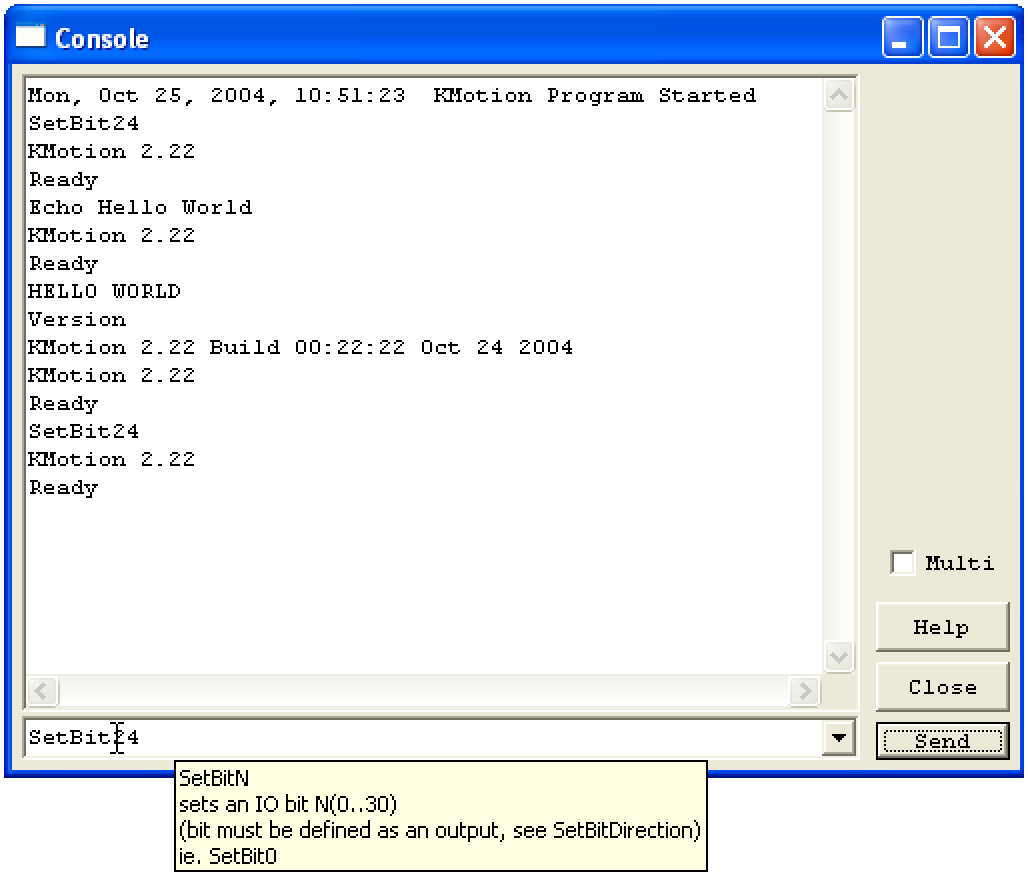
Example of data file follows:

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):



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.

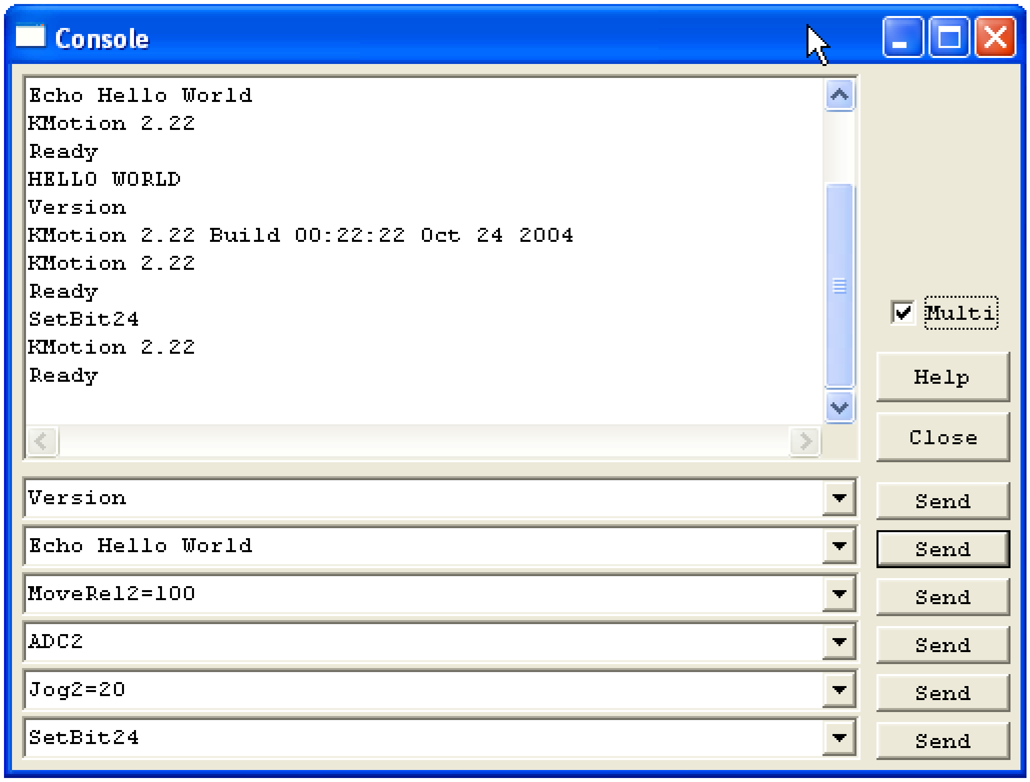
Selecting the 

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.

Multiple commands may be entered on a single line by separating the commands with a semicolon. For Example:

SetBit46;SetBit47

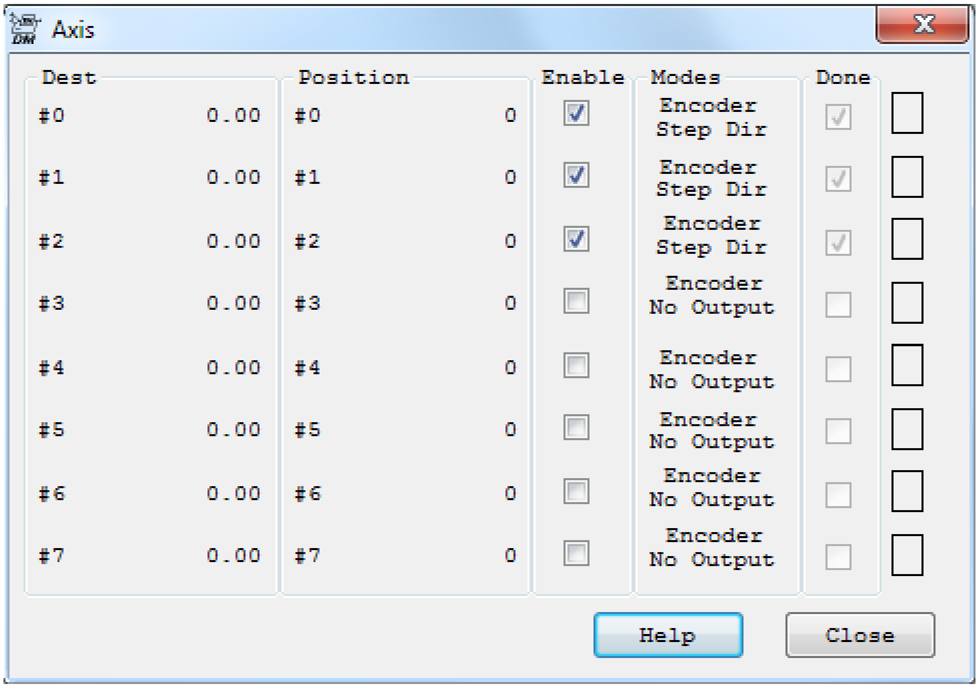
Inject<N> <F> <A>
InputChan<M> <N>=<C>
InputGain<M> <N>=<G>
InputMode<N>=<M>
InputOffset<M> <N>=<O>
InvDistPerCycle<N>=<X>
Jerk<N>=<J>
Jog<N>=<V>
Kill<N>
Lead<N>=<M>
LimitSwitch<N>=<H>
Linear <X0> <Y0> <Z0> <A0> <B0>
<C0>
<X1> <Y1> <Z1> <A1> <B1> <C1>
<a> <b> <c> <d> <tF>
LinearHex <X0> <Y0> <Z0> <A0>
<B0> <C0>
<X1> <Y1> <Z1> <A1> <B1> <C1>
<a> <b> <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>
MoveXYZABC <X> <Y> <Z>
<A><B><C>
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 ***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/Kogna.

Kogna Summary

16-Axis, DSP/FPGA-based Motion Controller

DynoMotion's Kogna card combines a 2.7 GFLOP DSP (TMS320C6748), FPGA, Ethernet, USB, and a PC-based development environment to create a versatile and programmable motion solution. Designed for up to 16-axes control, Kogna provides advanced control for torque, speed, and position for any mix of stepper, DC brushless, and DC brush motors. Kogna uses flash memory to store and run multiple-thread compiled C code on a 2.7 GFLOP processor with native 64-bit floating point support for stand-alone operation. A PC connected with an Ethernet 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 Kogna card via many languages - Visual C++, Visual C#, .NET etc. 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 Kogna packs a lot of IO into its 5.0 x 3.5 in package. Kogna offers 130 I/O bits, shared between dedicated IO and user-defined I/O. In addition analog inputs and outputs.

Kogna - Quick Start

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.

Remove the Kogna 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.**

Kogna 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:* Kogna 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 (***Kogna*** is checking if the Host is requesting a Flash Recovery during this time), and then the LEDs should remain steady.

Now connect an Ethernet cable from ***Kogna*** to your PC and run KMotion.exe. By default KMotion.exe uses any KFLOP or Kogna found by searching Ethernet or USB. If only one Kogna board is connected on the same network as the PC (or only one KFLOP on the PC's USB ports) connection should be automatic. Otherwise see *Using Multiple Boards*.

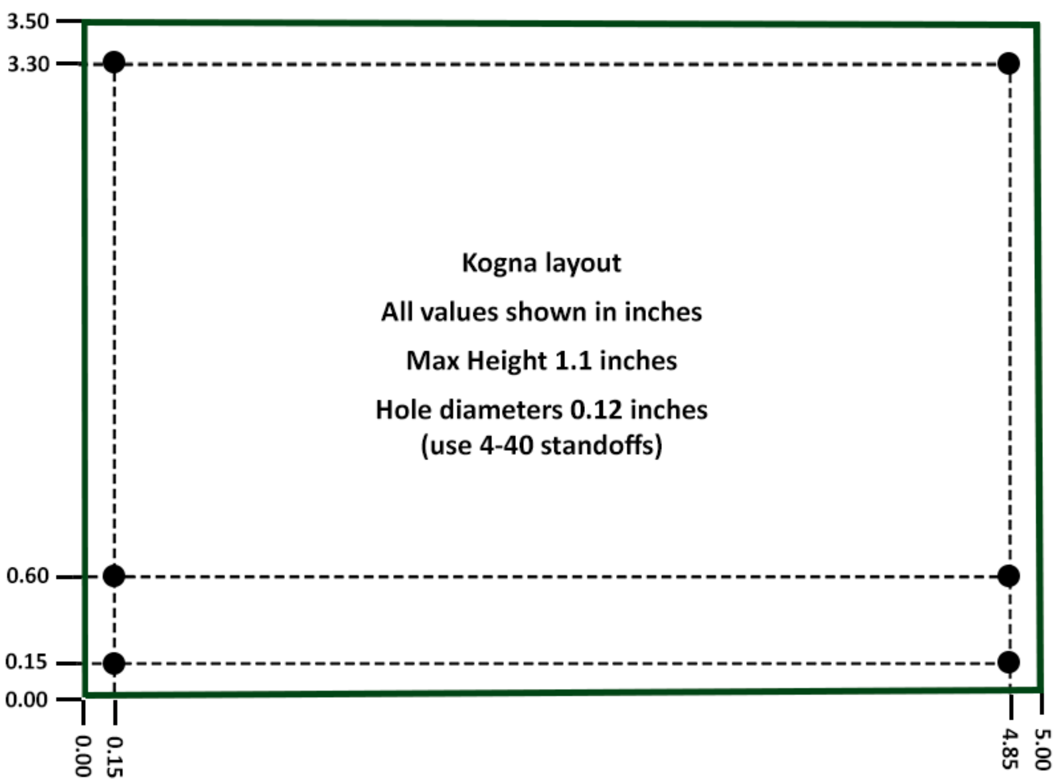
Kogna Hardware Specification

Function	Parameter	Specification
Processor	CPU Memory	TMS320C67-456MHz DSP 2.7 GFLOP 32/64-Bit Native Floating Point FLASH 512 MBytes SDRAM 128 Mbytes
Interface	Host	Ethernet 100Mbps
Connectors	JP7 I/O General Purpose JP5 I/O Com JP4 Aux#0 IO JP6 Aux#1 IO J14 Aux#2 IO J14 Ethernet J3 USB JP8 Differential Inputs / DACs JP10 Differential Outputs / ADCs / Hi-Res PWMs JP12 SPI / RS422/485 / EX_IO / Opto Relay Drivers JP14 - Differential Inputs / Differential Ouputs System Power	26 pin Header 8 pin RJ45 16 pin Header 16 pin Header 16 pin Header RJ45 Micro USB 40 pin Header 40 pin Header 40 pin Header 50 pin Header Molex 4-pin (Disk drive type)
Servo Loop	Sample Rate Compensation Feed Forward	90µs PID + (3) IIR bi-quad Stages/Axis Acceleration + Velocity
Axis	Number Type	16 MicroStep/Servo/Brush/Brushless/StepDirection
Logic Supply	Voltage Max Current Typical Current	+5V ±5% 2.5A 0.35 A
Analog Supplies	+15V on-board Generator -15V on-board Generator	1W 1W
User I/O	Digital - 3.3V LVTTL Digital - Differential Outputs Digital - Differential Inputs 24V Opto-Isolated Relay Drivers A/B Quadrature Encoder Inputs Step/Dir Pulse Generators PWM 8-bit Hi-Res PWM 16-bit DACs ADCs SPI I2C RS422/485	70 multipurpose Gen Purpose LVTTL (24 are 5V Tolerant) 24 24 4 (20) 5 MCounts/sec (16) 2.5MHz Pulse rate (8) 16.7MHz Clock rate 60KHz or prescaled by 1-256 (4) 228MHz Clock rate Programmable Period (8) +/-10V 12-bit 90us Update (4) 0-5V 10.5-bit 1620ms Update 76MHz Clock rate max 400KHz Clock rate max 1.5Mbaud max
Environment	Operating Temperature Storage Temperature Humidity	0-40° C 0-40° C 20-90% Relative Humidity, non-condensing
Dimensions	Length Width Height	3.5 inches (89mm) 5.0 inches (127 mm) 0.75inches (19 mm)

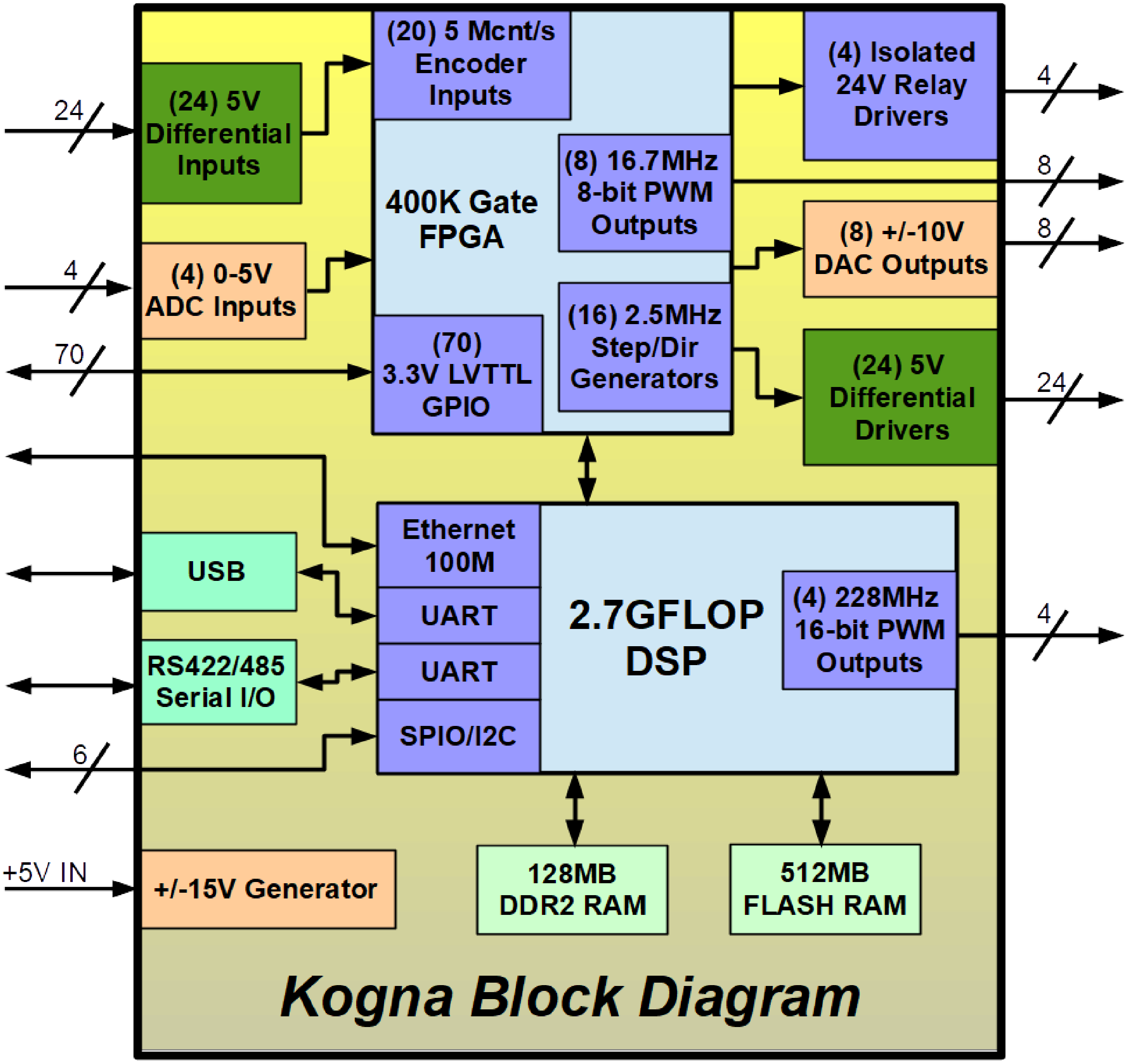
Kogna Software Specification

Function	Parameter	Specification
User Programs	Language Number concurrent Stand alone mode	C 7 Yes
Host Requirements	OS Interface	MS Windows™ 7 through 11
Interface Library	Multi-Thread Multi-Process Multi-Board MS Windows™ VC++ MS Windows™ VB MS Windows™ C# MS Windows™ .NET	Yes Yes Yes Supported Supported Supported Supported
C Compiler	TCC67	Included
G Code	Interpreter	Included
Script Language	ASCII Commands	Included
Trajectory Planner	Coordinated Motion	6 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

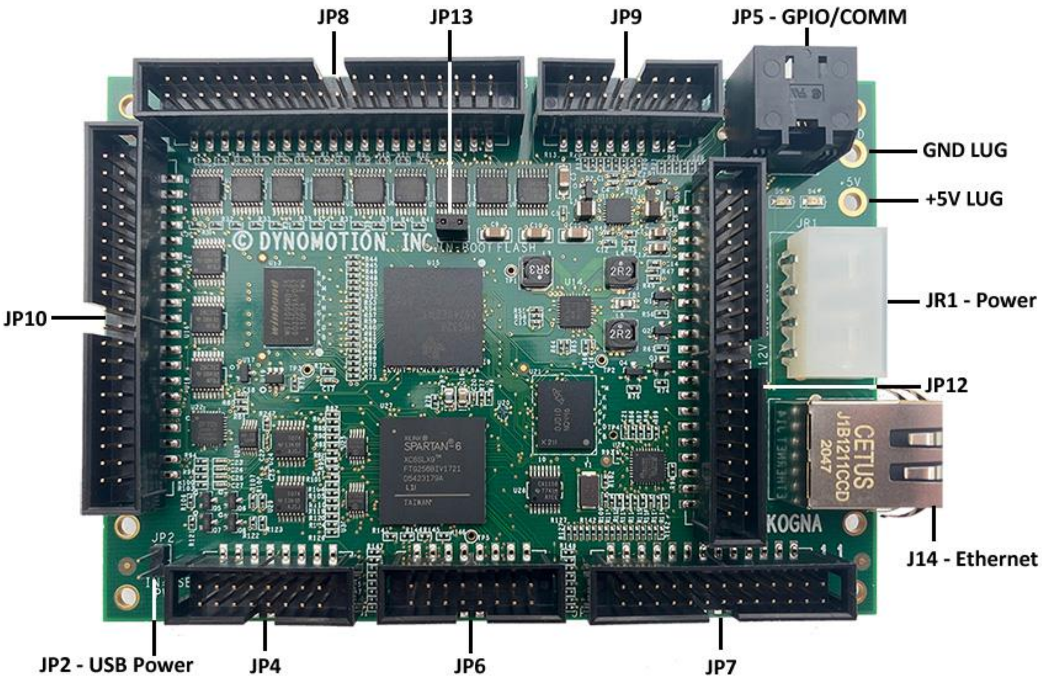
Kogna – Board Layout

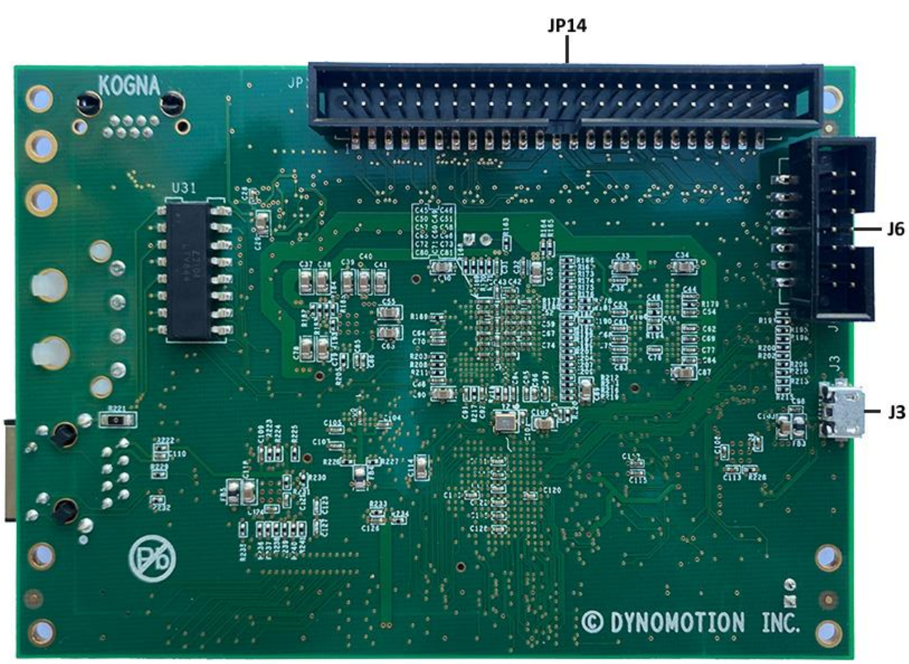


Kogna – Block Diagram



Kogna – Connector Pin outs





KFLOP Equivalent Connectors J6, JR1, JP2, JP4, JP5, JP6, and JP7

These connectors have equivalent pin outs and functionality as the connectors on KFLOP. See the KFLOP Connector documentation for these connectors here.

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

JP2 - JTAG

JP7 - Digital IO

JP4 - Aux Connector #0

JP6 - Aux Connector #1

JP5 - GPIO / LV Differential Connector

Kogna Connectors - J14, J3, JP8, JP9, JP10, JP12, JP14, Power Lugs, JP2, JP3

J14 Ethernet

J3 USB / Virtual COM Port

JP8 - Differential Inputs / DACs

JP9 - Aux Connector #2

JP10 Differential Outputs / ADCS / Hi-Res PWMs

JP12 SPI / RS422/485 / EX_IO / Opto Relay Drivers

JP14 - Differential Inputs / Differential Outputs

Kogna Power Lugs

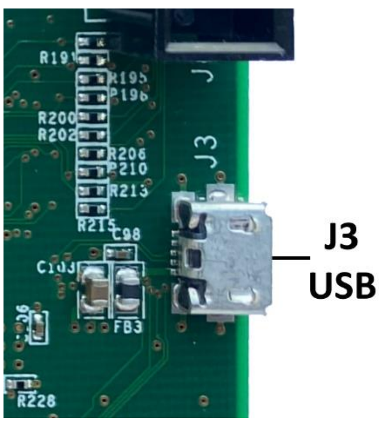
JP2 USB Power Jumper

JP13 Flash Boot Jumper

J14 Ethernet

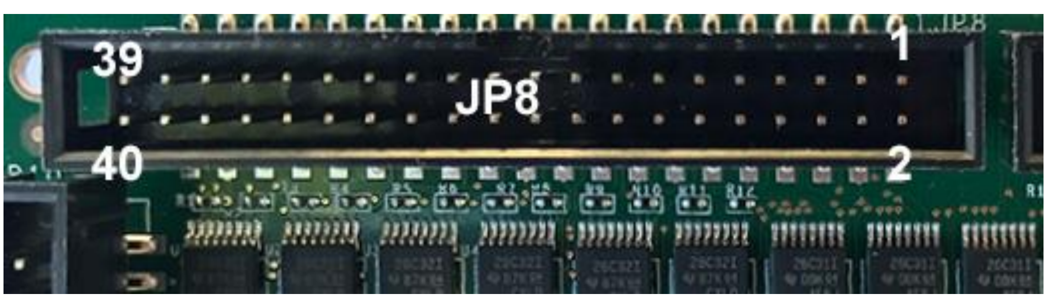
J14 is a standard RJ45 100M Ethernet connector. The metal case is connected to Kogna GND.

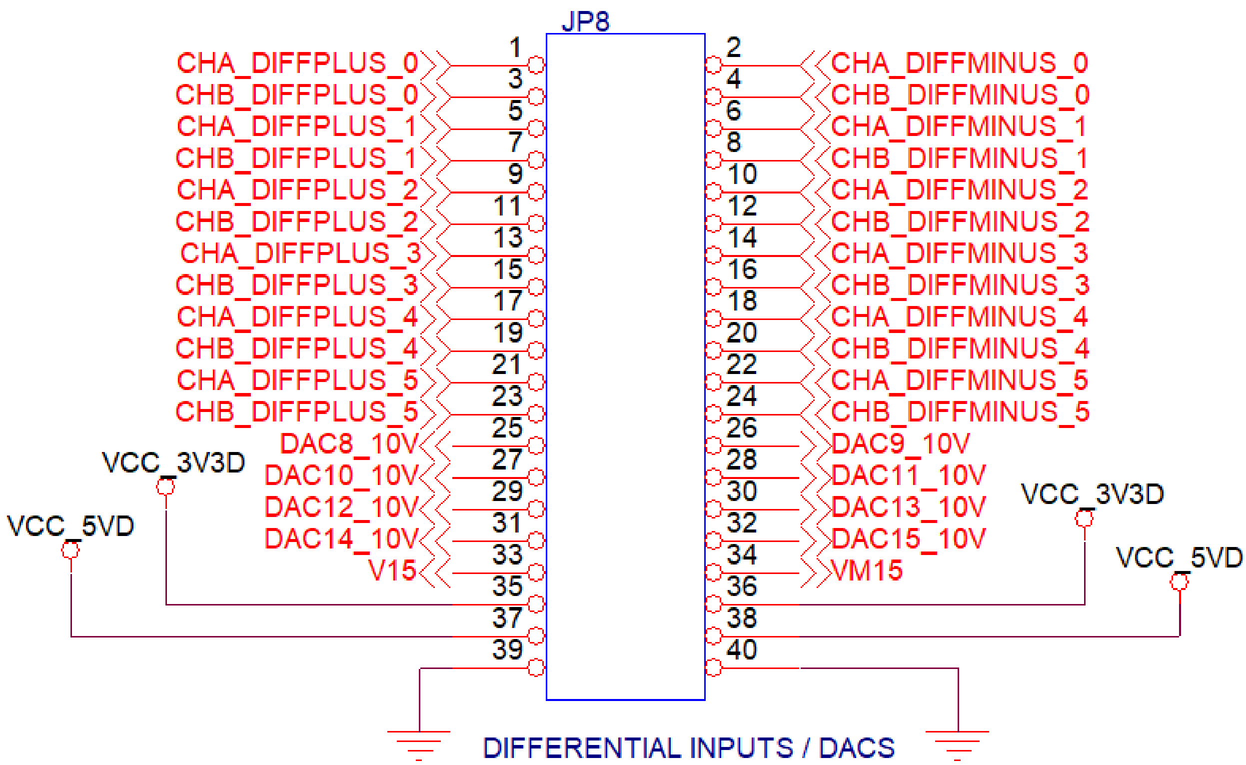
J3 USB / Virtual COM Port



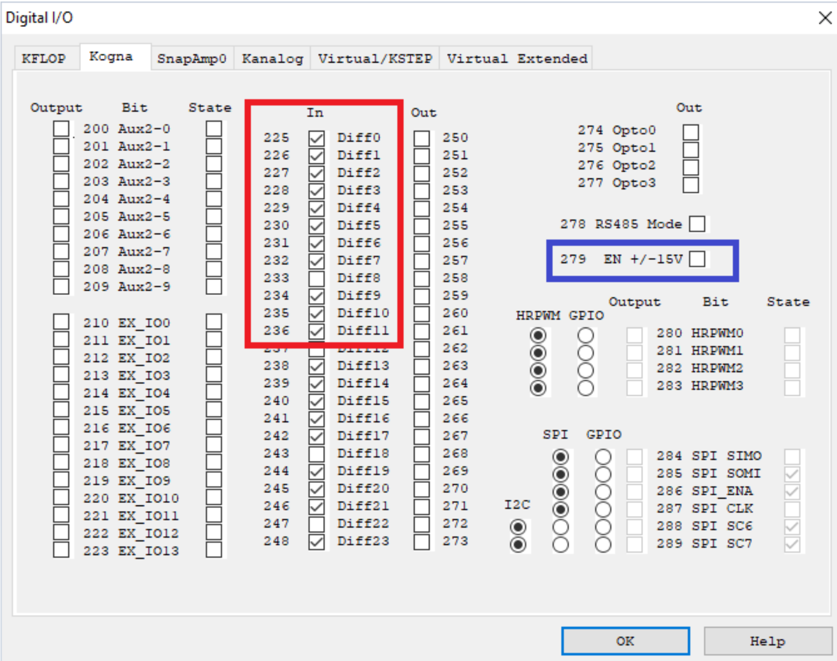
Standard Micro USB Connector. Shield is connected to Kogna GND. See here for more information.

JP8 - Differential Inputs / DACs

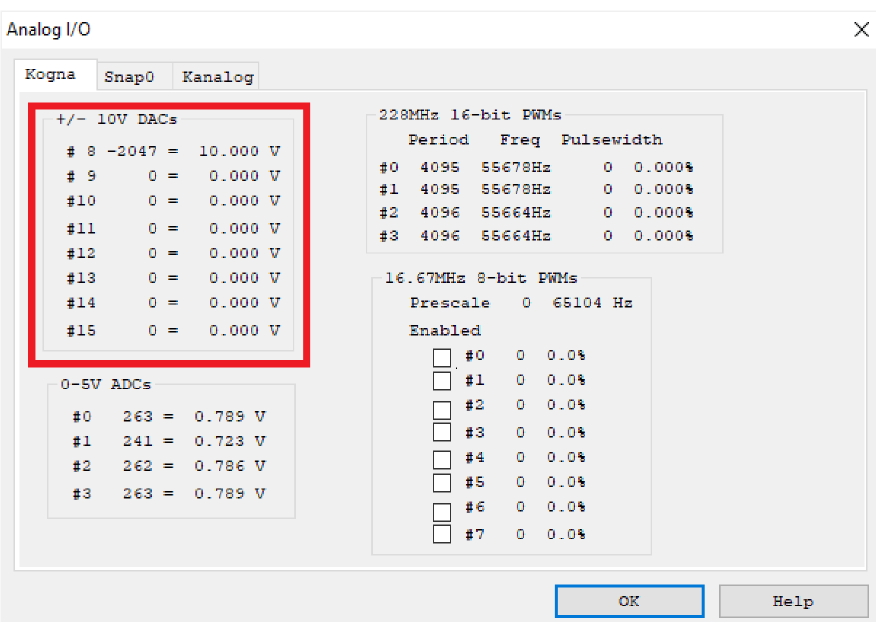




JP8 contains the first half of Kogna's 5V differential inputs (JP14 contains the other half). Enough for 6 A/B quadrature encoders inputs. See Kogna's Encoders for more information. Each encoder has an an A and B input where each A and B input has a + and - input pin. When the + pin is at a higher voltage than the - pin then the input is high. When the + pin is at a lower voltage than the - pin the input is low. The + and - pins have a moderate termination resistance of 470 Ohms. Both the + and - pins must be driven for the input to work properly (neither can be left unconnected). The differential inputs may also read as 12 General Inputs as IO Bits 225 - 236 and displayed on the Digital IO Screen as shown in red below:



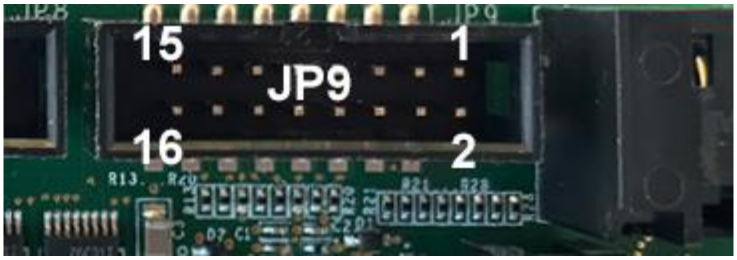
JP8 also contains Kogna's 8 +/-10V Analog DACs referenced as DAC Channels #8 - #15 (#0 - #7 refer to Kanalog DACs which are available with the Kanalog Option Board). Kogna contains an on-board 5V to +/-15V supply generator which is controlled by output Bit 279 and must be turned on in order for the DAC Outputs to be driven. See Digital IO Screen above marked in blue. Note the DAC Outputs will be indeterminate and non-zero until the +/-15V is activated. The current DAC settings are displayed on the Analog Status Screen shown below marked in red. The DACs have 12-bit resolution where +/-2047 counts correspond to +/-10V. All DACs update every Servo Sample time of 90us.



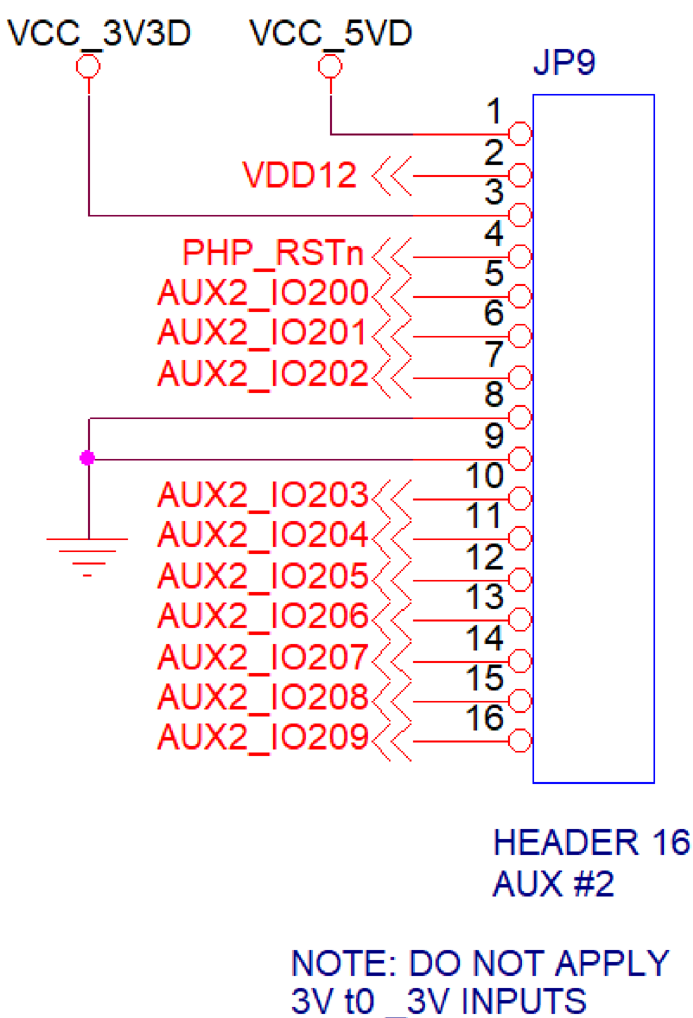
JP8 Pins	Bit	Description
1,2	225	CHA0 - Encoder #8
3,4	226	CHB0 - Encoder #8
5,6	227	CHA1 - Encoder #9
7,8	228	CHB1 - Encoder #9
9,10	229	CHA2 - Encoder #10
11,12	230	CHB2 - Encoder #10
13,14	231	CHA3 - Encoder #11
15,16	232	CHB3 - Encoder #11
17,18	233	CHA4 - Encoder #12
19,20	234	CHB4 - Encoder #12
21,22	235	CHA5 - Encoder #13
23,24	236	CHB5 - Encoder #14
JP8 Pin	**DAC**	**Description**
25	8	+/-10V Analog DAC
26	9	+/-10V Analog DAC
27	10	+/-10V Analog DAC
28	11	+/-10V Analog DAC

29	12	+/-10V Analog DAC
30	13	+/-10V Analog DAC
31	14	+/-10V Analog DAC
32	15	+/-10V Analog DAC
JP8 Pin	**Supply**	**Description**
33	+15V	On-board generated +15V
34	-15V	On-board generated +15V
35	+3.3V	On-board generated 3.3V
36	+3.3V	On-board generated 3.3V
37	+5V	+5V Supply
38	+5V	+5V Supply
39	0V	Kogna GND
40	0V	Kogna GND

JP9 - Aux Connector #2

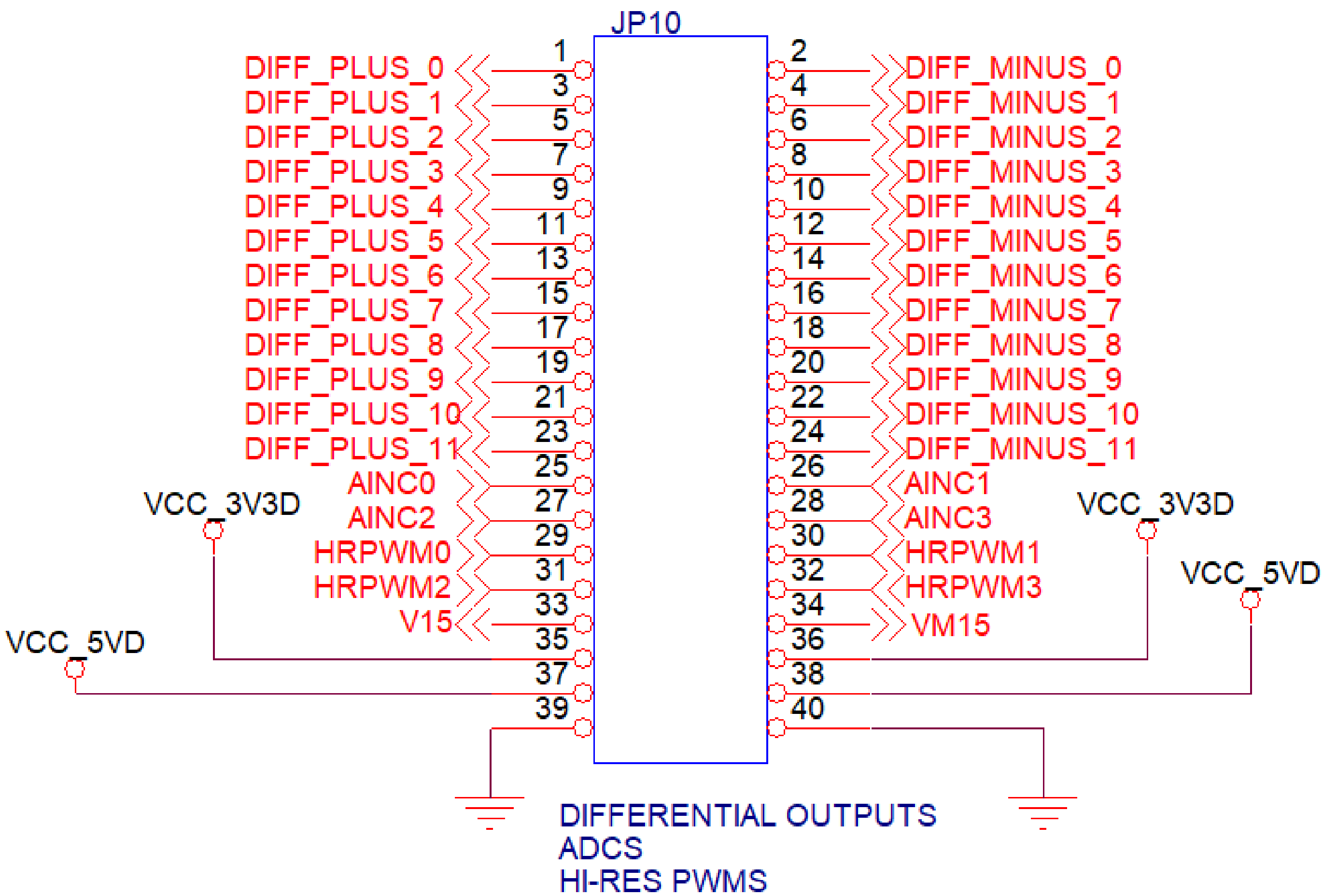


Kogna adds a 3rd 16-pin Aux Connector that is electrically equivalent to KFLOP's Aux #0 and Aux #1 connectors. The Auxiliary connector supplies power, reset, and 10 digital I/O (LVTTL 3.3V only). These digital I/O may be used for general purpose use and are referenced as IO Bits 200 - 209. The first 8 IO (IO200 - IO207) contain 150 Ohm termination resistors (pull downs).



JP9 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	AUX2 IO200	Gen Purpose LVTTL I/O (3.3V Only)
6	AUX2 IO201	Gen Purpose LVTTL I/O (3.3V Only)
7	AUX2 IO202	Gen Purpose LVTTL I/O (3.3V Only)
8	GND	Digital Ground
9	GND	Digital Ground
10	AUX2 IO203	Gen Purpose LVTTL I/O (3.3V Only)
11	AUX2 IO204	Gen Purpose LVTTL I/O (3.3V Only)
12	AUX2 IO205	Gen Purpose LVTTL I/O (3.3V Only)
13	AUX2 IO206	Gen Purpose LVTTL I/O (3.3V Only)
14	AUX2 IO207	Gen Purpose LVTTL I/O (3.3V Only)
15	AUX2 IO208	Gen Purpose LVTTL I/O (3.3V Only)
16	AUX2 IO209	Gen Purpose LVTTL I/O (3.3V Only)

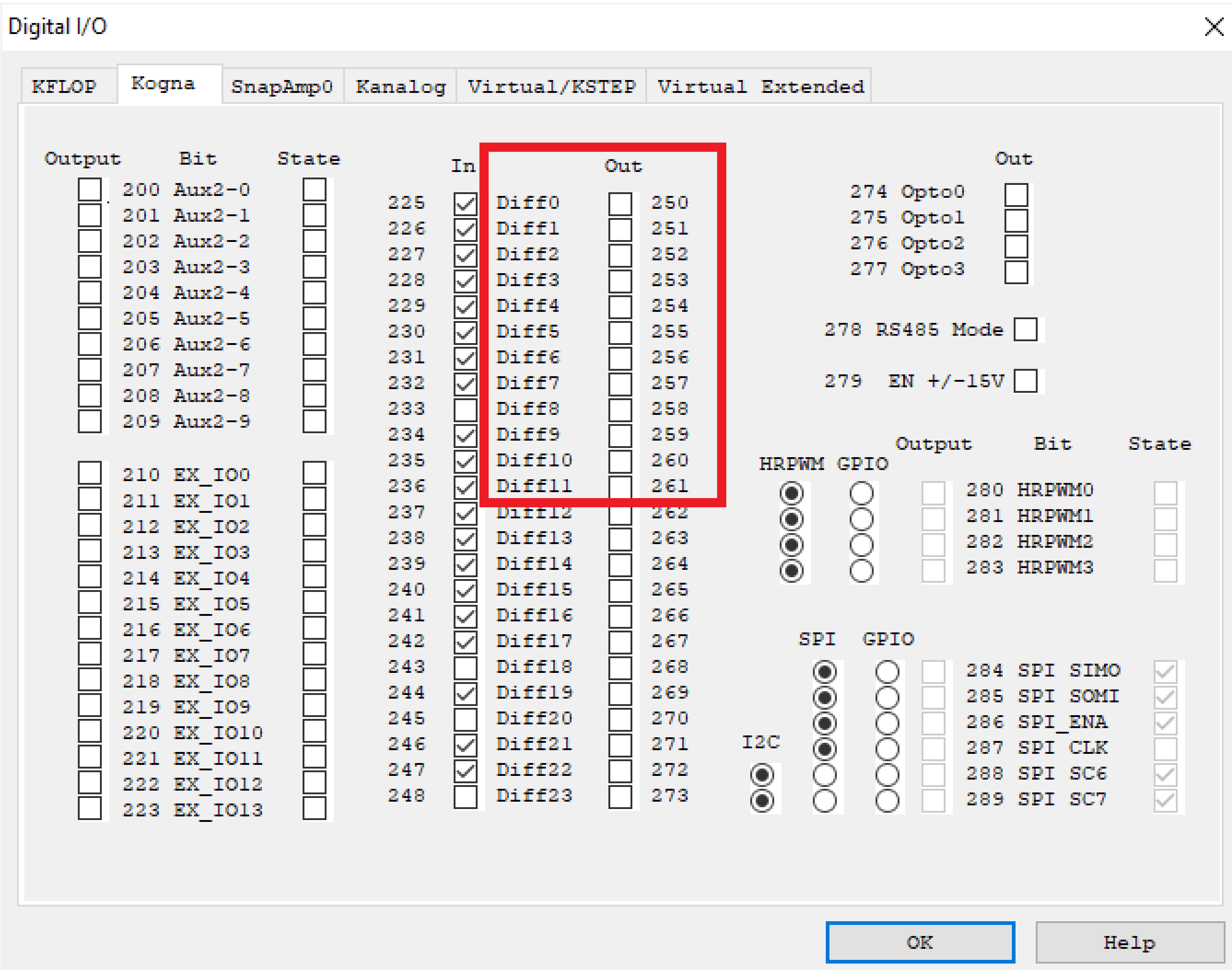
JP10 Differential Outputs / ADCs / Hi-Res PWMs



JP10 contains the first half of Kogna's 5V differential outputs (JP14 contains the other half). Enough for 6 Axes of Step/Direction being driven with a Differential Driver. Step/Direction Generators #0 - #5 can be configured to drive pins on JP10. See [Kogna's Step/Direction](#) for more information. Each Step/Direction Generator has a Step and Direction Output where each Step and Direction output each has a + and - output pin. When the signal is high the + pin is driven near 5V and the - pin is driven near 0V. When the signal is low the + pin is driven near 0V and the - pin is driven near 5V.

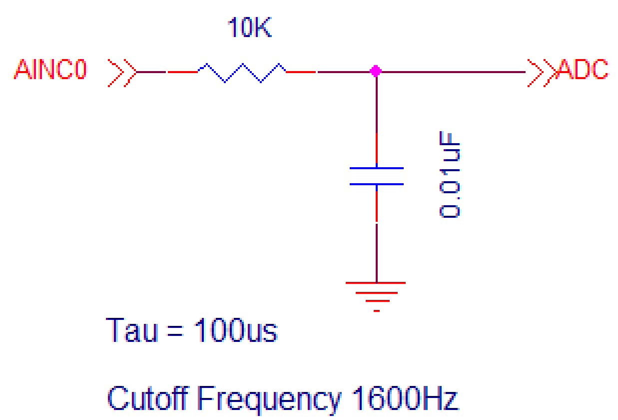
To drive single-ended (non-differential) inputs one of the signals may be used with the other left unconnected. In this case the pin is used as a +5V TTL output. To invert the polarity use the - pin rather than the + pin.

If the associated Step/Direction channel is not enabled the differential outputs may be used as 12 General Outputs as IO Bits 250 - 261. The current state can be toggled from the Digital IO Screen as well as the last commanded state displayed as shown in red below:

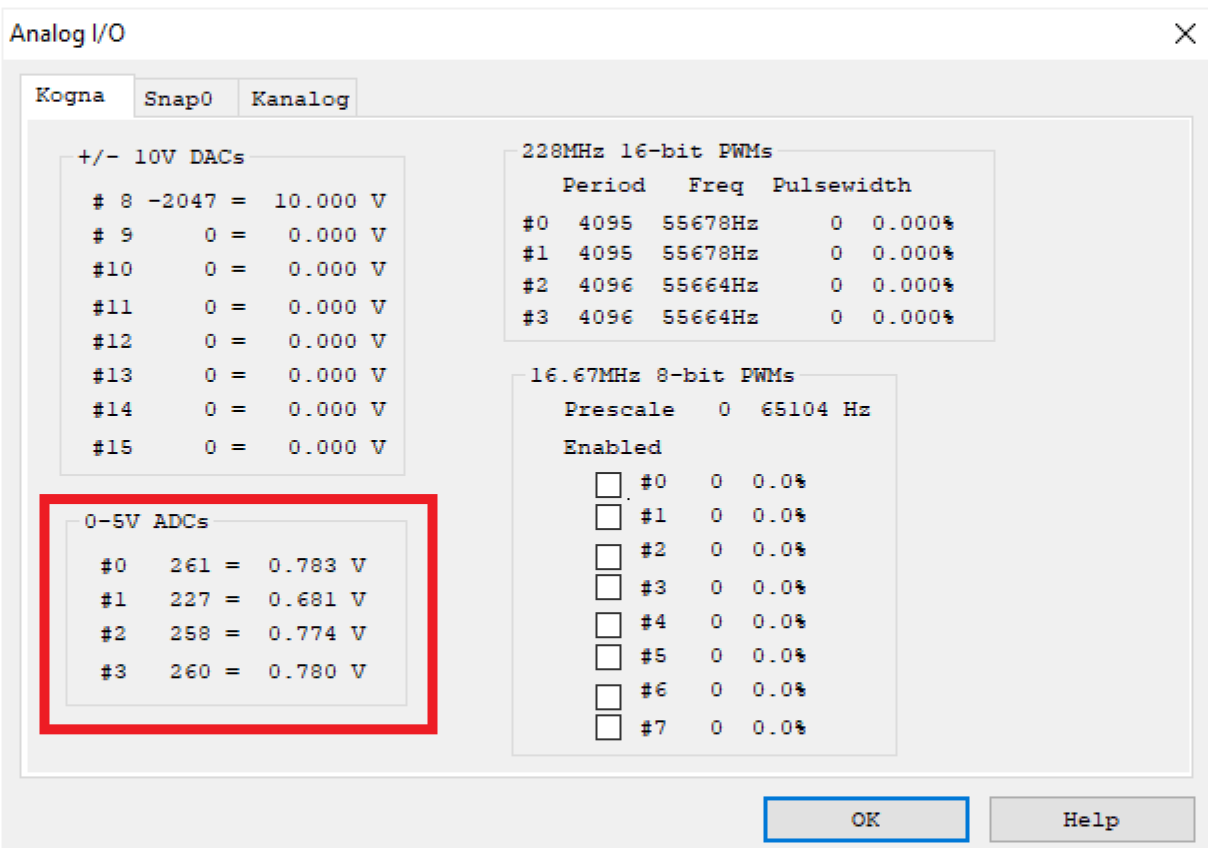


JP10 also contains Kogna's 4 0-5V Analog ADCs referenced as ADC Channels #8 - #11 (#0 - #7 refer to Kanalog ADCs which are available with the Kanalog Option Board).

Each ADC input has a low pass filter with cutoff frequency at 1600Hz. DC Input Impedance > 200K Ohms.



The current ADC readings are displayed on the Analog Status Screen shown below marked in red. The DACs have ~11-bit resolution where 2047 counts correspond to 6.144V, however the inputs have protection diodes to the +5V supply and can not be driven above ~5.5V. All ADCs update every 1620us.



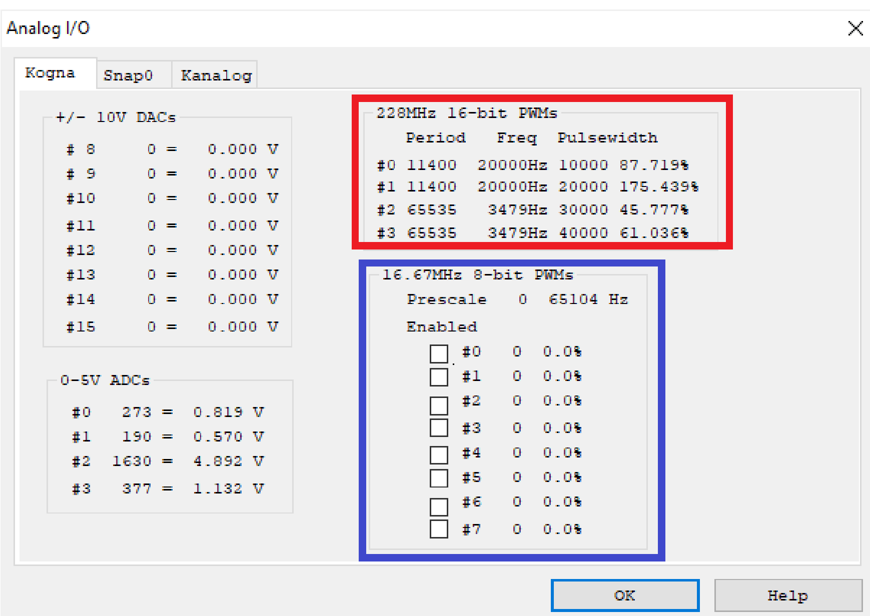
JP10 also contains Kogna's 4 High-Resolution PWMS. Because these are clocked at 228MHz they are able operate at 12-bits of resolution at 55.7KHz. The example below shows how the connector pins can be set to HRPWM mode, and how the Period and and Pulse length can be set. HRPWMs #0 and #1 have a common binary prescaler (divide by 2 ^ N) and a common

Period. HRPWMs #2 and #3 do not have a prescale and have independent Period registers. This example is included as HRPWM Example.c.

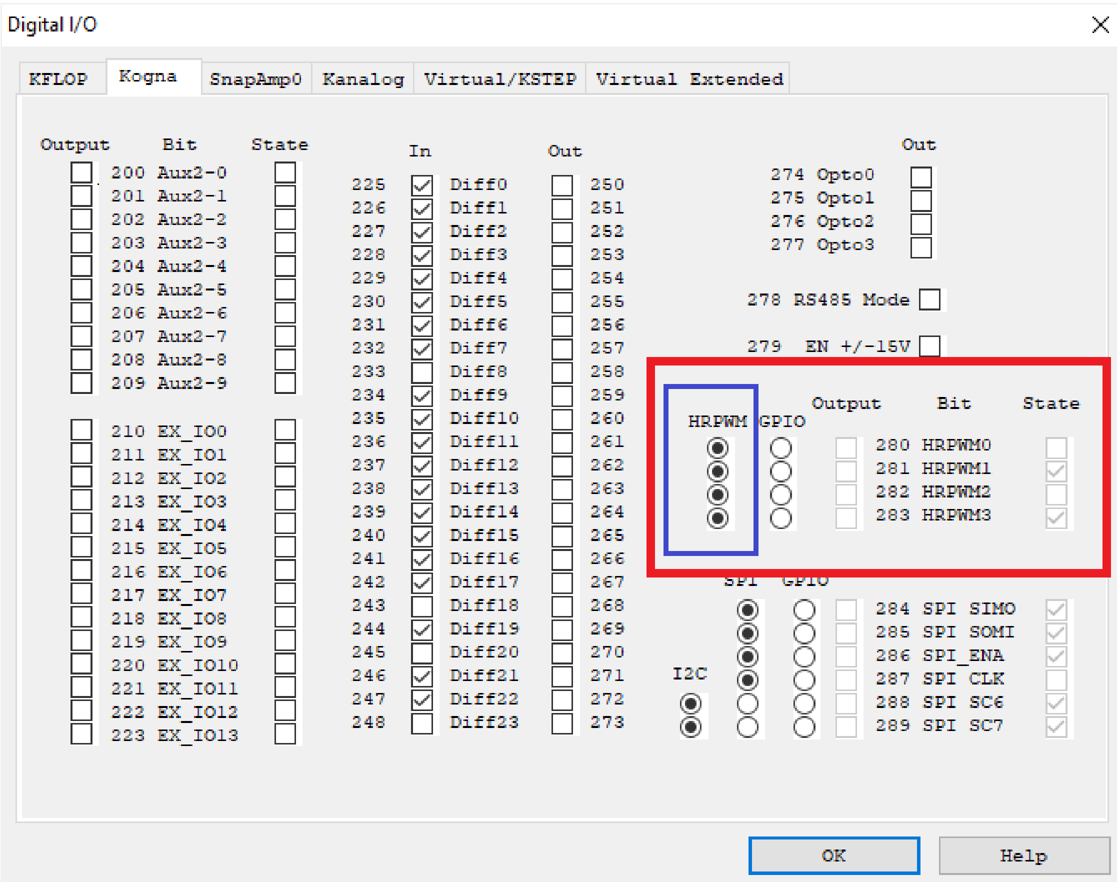
```
#include "KMotionDef.h"

main()
{
    int i;

    for (i=0; i<4; i++)
        HRPWM_SetMode(i,0);  // switch from GPIO to HRPWM Mode

    HRPWM_01_PRESCALE_POWER(0);
    HRPWMPERIOD01 = 11400;    // 20KHz 13+ bits resolution
    HRPWM0= 10000;            // Set HRPWM 0 Period
    HRPWM1= 20000;            // Set HRPWM 1 Period

    HRPWMPERIOD2 = 65535;     // 3.5KHz 16 bits resolution
    HRPWM2 = 30000;           // Set HRPWM 2 Period

    HRPWMPERIOD3 = 65535;     // 3.5KHz 16 bits resolution
    HRPWM3 = 40000;           // Set HRPWM 3 Period
}
```

The current HRPWM settings can be observed on the Analog Status Screen as shown in red below.

The original (8) 8-bit 16.67 MHz PWMs, Prescale, and enables are also displayed. When enabled the PWM drives the JP6 Pin otherwise it can be used as a GPIO pins (IO Bits 26 - 33). See Analog Status Screen as shown in blue below.



The current HRPWM state is displayed on the Digital IO Screen as shown below in red. These 4 3.3V LVTTL connector pins can be configured as HRPWMs or GPIO (inputs or outputs) IO Bits 280-283. In the Screen below they are shown to be HRPWM mode as shown in blue. The GPIO have fast access times as they are directly connected to Kogna's DSP.

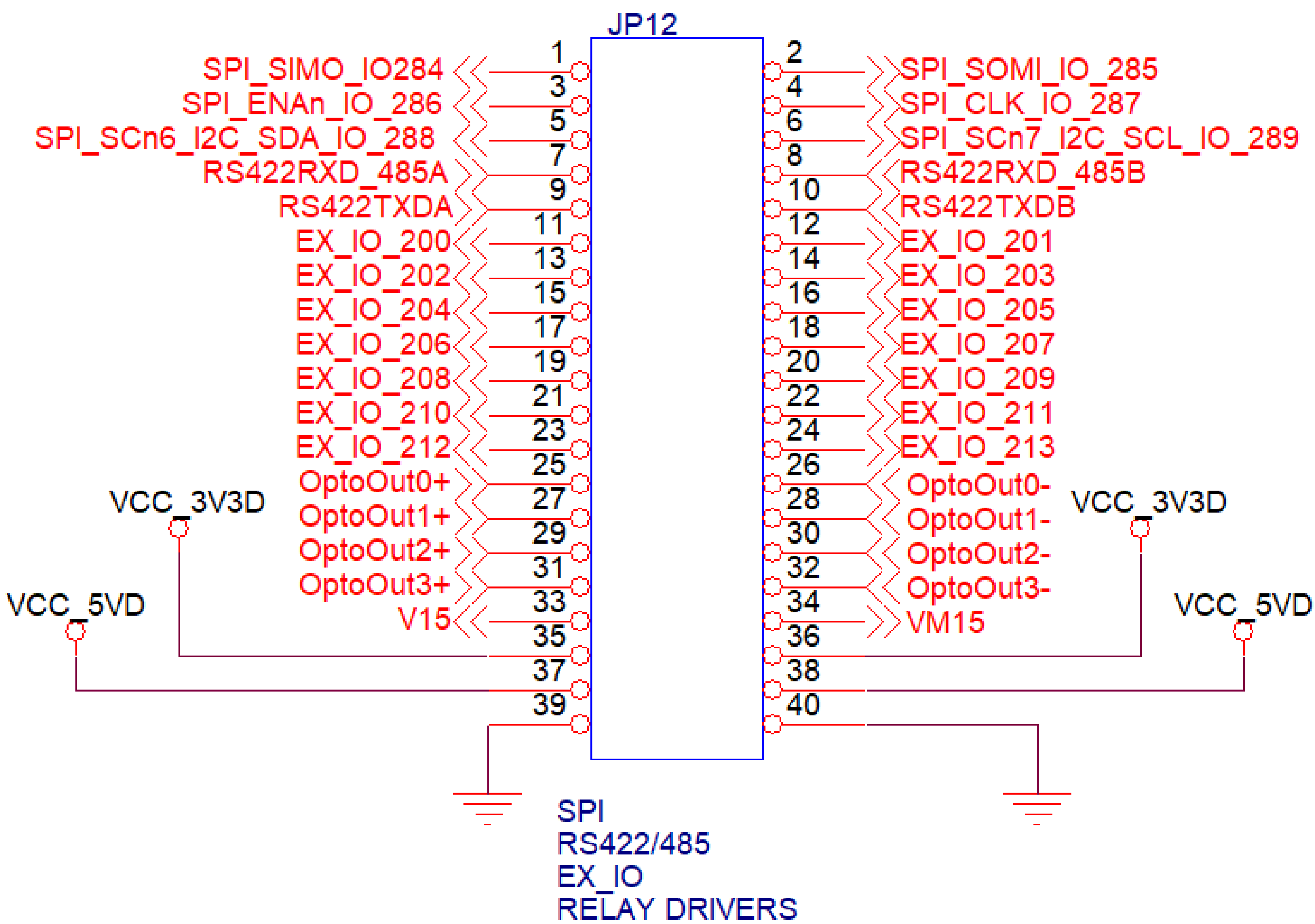


JP10 Pin Descriptions

JP10 Pins	Bit	Description
1,2	250	Step #0
3,4	251	Direction #0
5,6	252	Step #1
7,8	253	Direction #1
9,10	254	Step #2
11,12	255	Direction #2
13,14	256	Step #3
15,16	257	Direction #3
17,18	258	Step #4
19,20	259	Direction #4
21,22	260	Step #5
23,24	261	Direction #5
JP10 Pin	**ADC**	**Description**
25	8	0-5V ADC
26	9	0-5V ADC

27	10	0-5V ADC
28	11	0-5V ADC
JP10 Pin	**PWM**	**Description**
29	0	Hi-Res PWM
30	1	Hi-Res PWM
31	2	Hi-Res PWM
32	3	Hi-Res PWM
JP10 Pin	**Supply**	**Description**
33	+15V	On-board generated +15V
34	-15V	On-board generated +15V
35	+3.3V	On-board generated 3.3V
36	+3.3V	On-board generated 3.3V
37	+5V	+5V Supply
38	+5V	+5V Supply
39	0V	Kogna GND
40	0V	Kogna GND

JP12 SPI / RS422/485 / EX_IO / OPTO RELAY DRIVERS



JP12 contains (6) 3.3V LVTTL SPI pins connected directly to Kogna's 2-16 bit hardware SPI 1 peripheral that can theoretically operate up to 76 MHz. 3 SPI Enable pins are provided. See the TI C6748 Technical Reference Manual for operation.

(2) pins can also operate as I2C Pins connected directly to Kogna's hardware I2C0 peripheral that can operate up to 400KHz.

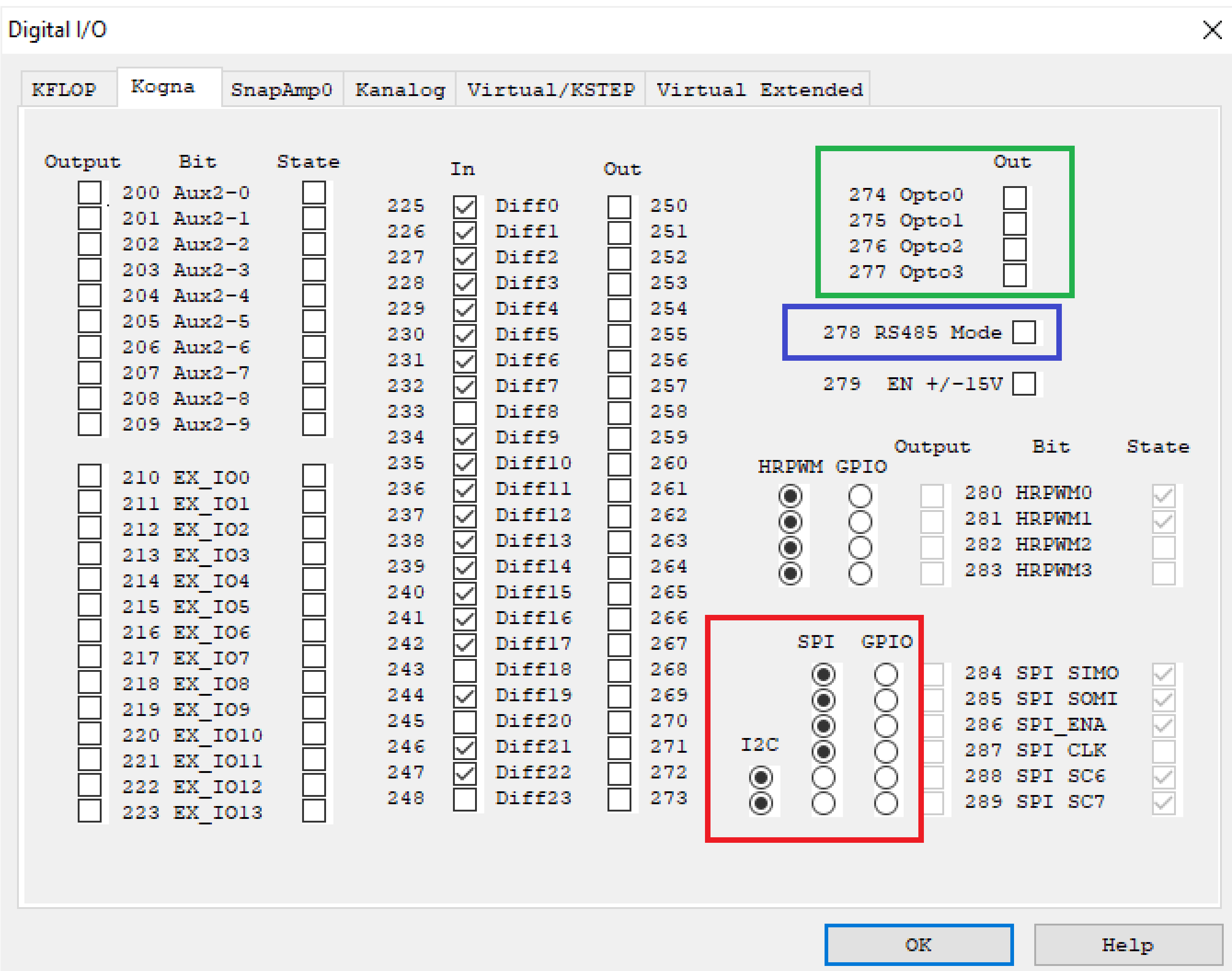
4 of the 6 pins may be used as SPI pins or GPIO pins. The GPIO pins are IO Bits 284 - 287.

2 of the 6 pins may be used as SPI pins or I2C pins or GPIO pins. The GPIO pins are IO Bits 288 - 287.

C Functions or Script Commands can be used to change the modes of the Pins.

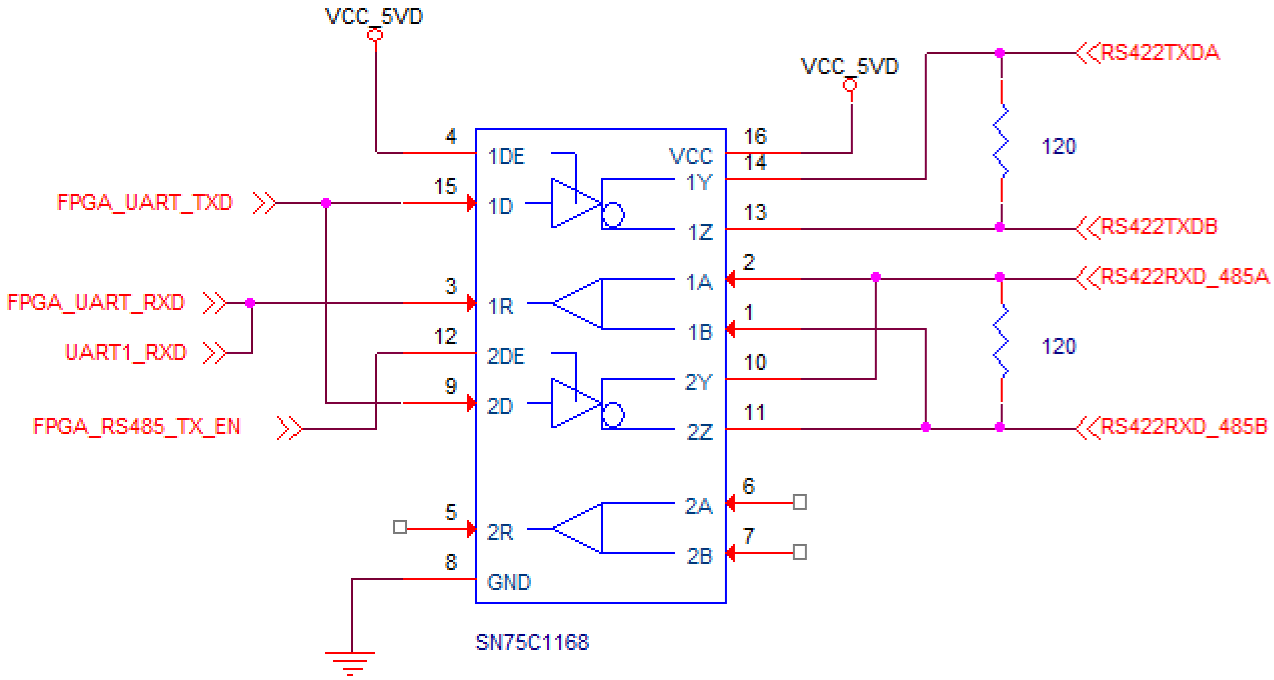
```
// Controls 6 DSP Pins for SPI/I2C or GPIO (last 2 can be I2C)

void SPI_SetMode(int chan, int GPIO);     // Set SPI mode as SPI or GPIO, 1=GPIO 0=SPI
2=I2C

int SPI_GetMode(int chan);                // Get SPI mode as SPI or GPIO or I2C, 1=GPIO
0=SPI 2=I2C
```

The current mode is displayed on the Digital IO Screens and can also be modified. See in red below.



JP12 also contains 5V differential signals interfaced to Kogna's hardware UART that operates up to 1.5Mbaud. 4 pins provide RS422/RS485 support with 5V differential Transmit and

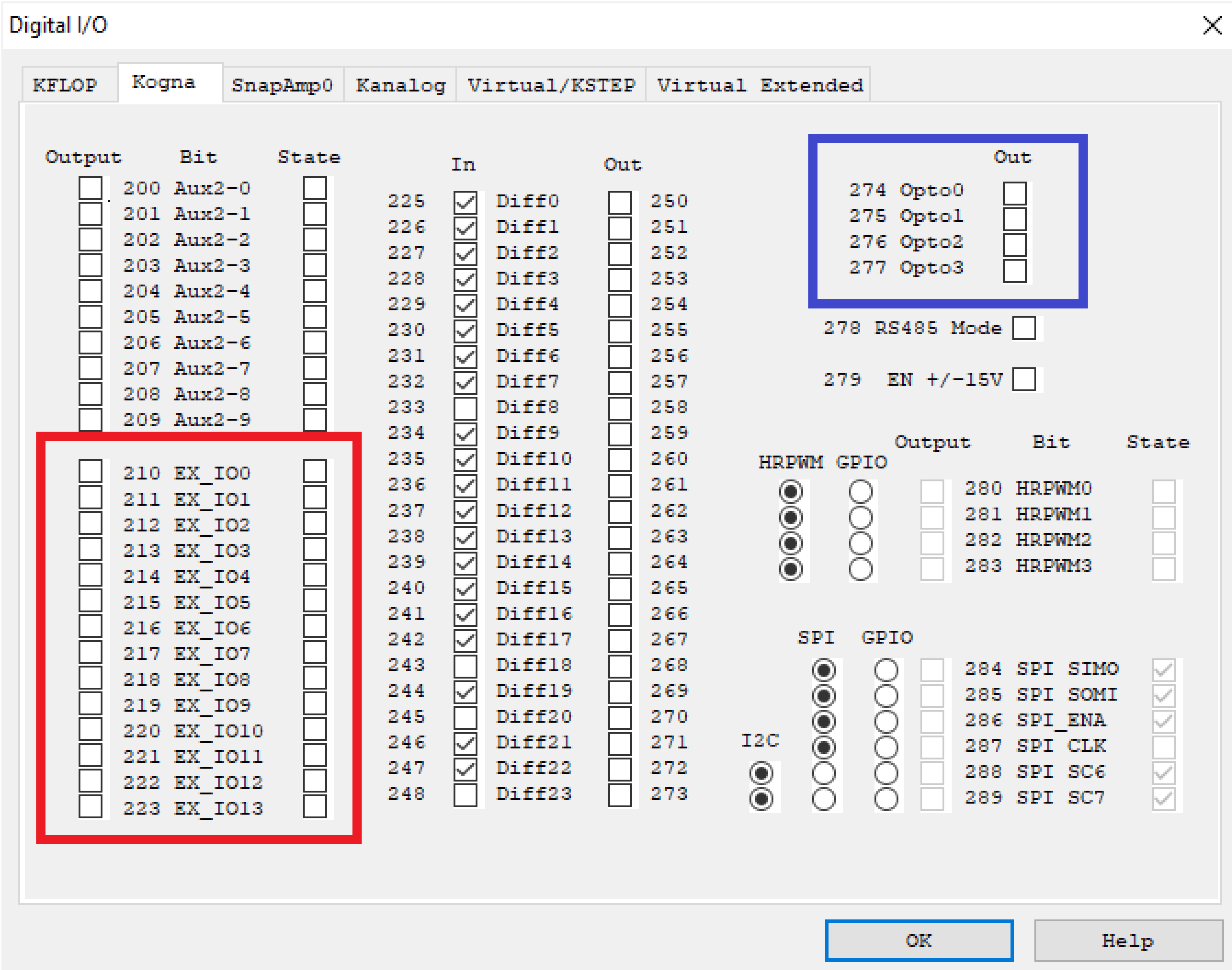
Receive signals. In RS422 mode 2 pins are used for the differential Transmit signal and 2 pins are used for the differential receive signal. In RS485 mode both Transmit and Receive operate on the same 2 pins (the other 2 pins are not used). Kogna's FPGA provides automatic output drive enable when in RS485 mode. Both differential pin pairs have 120 Ohm termination. See the circuit diagram below.



IO Bit 278 on selects RS485 mode and off selects RS422 mode. See the Digital IO Screen above in blue. The C Function below can be used to configure the UART for baud rate, word size, parity, and to select RS422 or RS485 mode. See the BufferedRS485Master.c and BufferedRS485 Slave Echo.c for an example of how 2 Kogna's might communicate over RS485.

```
// Rate to 1.5MBaud, nbits 5-8, Enable Parity adds a bit, mode 1=RS485 0-RS422

void RS422_SetBaudRate(int Rate, int nBits, int EnableParity, int ParityOdd, int RS485_M
ode);
```

JP12 also contains (14) 3.3 V LVTTL Extended GPIO Pins EX_IO 0 - 13 and as IO Bits 200 - 213. The current IO direction and State is displayed on the Digital IO Screen. See in red below.

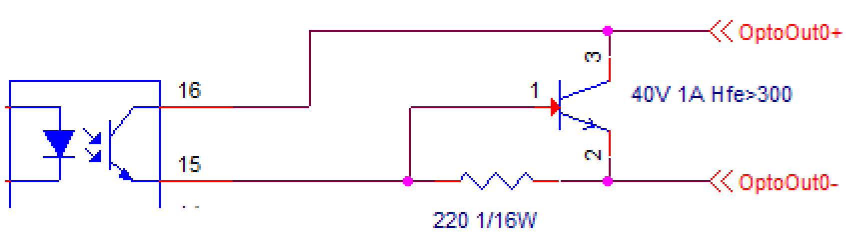


JP12 also contains (4) Opto Isolated Relay Driver Outputs rated for up to 30V @ 0.2A. The Optos are controlled by IO Bits 274-277. Current State can be displayed and controlled using the Digital IO Screen, See above in blue.

The circuit below is used.

For load currents < 0.5ma the on state voltage drop is < 0.35V

For load currents < 0.2A the on state voltage drop is < 0.8V



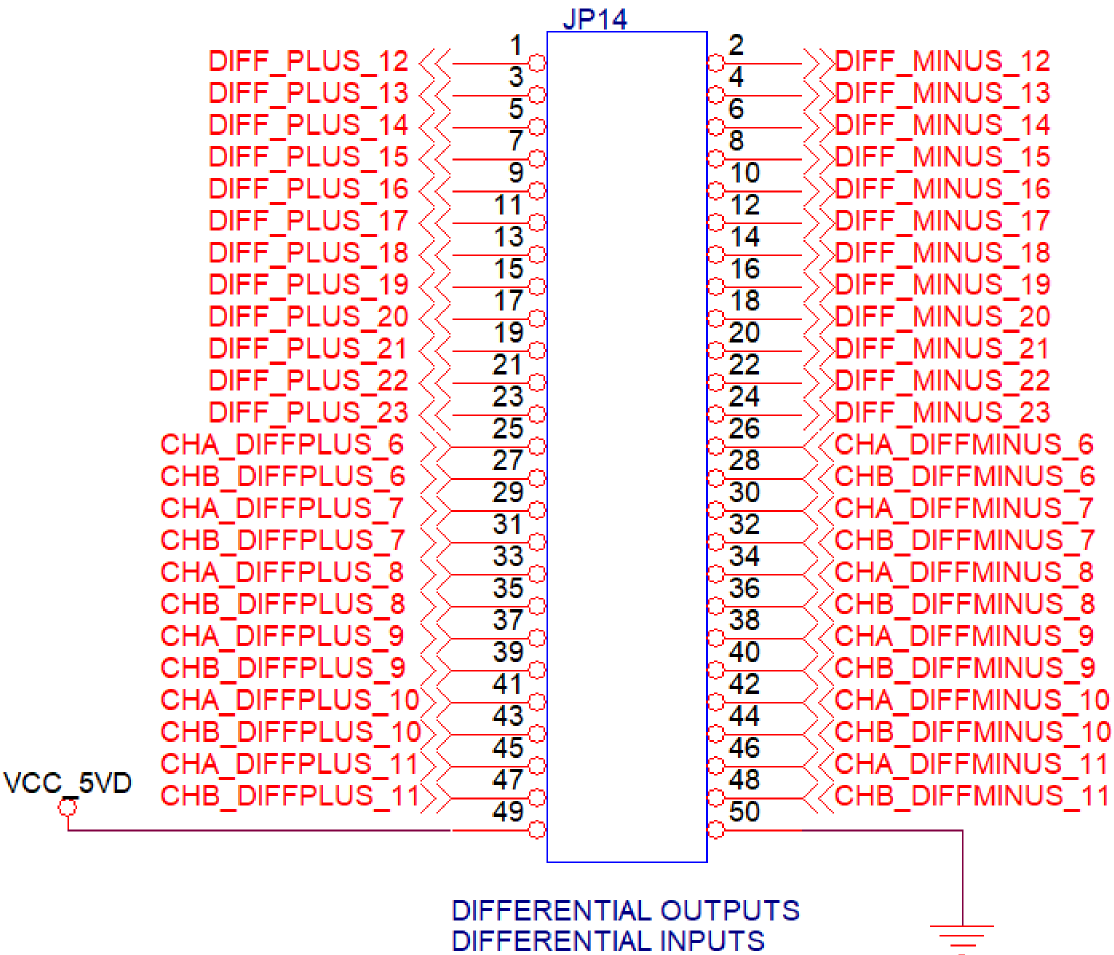
JP12 Pin Descriptions

JP12 Pin	Bit	SPI / I2C
1	284	SPI SIMO
2	285	SPI SOMI
3	286	SPI ENAn
4	287	SPI CLK
5	288	SPI SCn6 / I2C SDA
6	289	SPI SCn7 / I2C SCL
JP12 Pin		**RS242/485**
7	---	RS422 RXD / 485 A
8	---	RS422 RXD / 485 B
9	---	RS422 TXD A
10	---	RS422 TXD B
JP12 Pin	**Bit**	**EX IO / Step/Direction**
11	210	Gen Purpose LVTTL I/O (3.3V Only) / Step 12
12	211	Gen Purpose LVTTL I/O (3.3V Only) / Direction 12
13	212	Gen Purpose LVTTL I/O (3.3V Only) / Step 13)

14	213	Gen Purpose LVTTL I/O ((3.3V Only) / Direction 13)
15	214	Gen Purpose LVTTL I/O ((3.3V Only) / Step 14)
16	215	Gen Purpose LVTTL I/O ((3.3V Only) / Direction 14
17	216	Gen Purpose LVTTL I/O ((3.3V Only) / Step 15
18	217	Gen Purpose LVTTL I/O ((3.3V Only) / Direction 15
19	218	Gen Purpose LVTTL I/O ((3.3V Only)
20	219	Gen Purpose LVTTL I/O ((3.3V Only)
21	220	Gen Purpose LVTTL I/O ((3.3V Only)
22	221	Gen Purpose LVTTL I/O ((3.3V Only)
23	222	Gen Purpose LVTTL I/O ((3.3V Only)
24	223	Gen Purpose LVTTL I/O ((3.3V Only)
JP12 Pins	**Bit**	**Opto Isolated Relay Drivers**
25, 26	274	Opto Out 0 +/-
27,28	275	Opto Out 1 +/-
29,30	276	Opto Out 2 +/-
31,32	277	Opto Out 3 +/-
JP12 Pin	**Supply**	**Description**

33	+15V	On-board generated +15V
34	-15V	On-board generated +15V
35	+3.3V	On-board generated 3.3V
36	+3.3V	On-board generated 3.3V
37	+5V	+5V Supply
38	+5V	+5V Supply
39	0V	Kogna GND
40	0V	Kogna GND

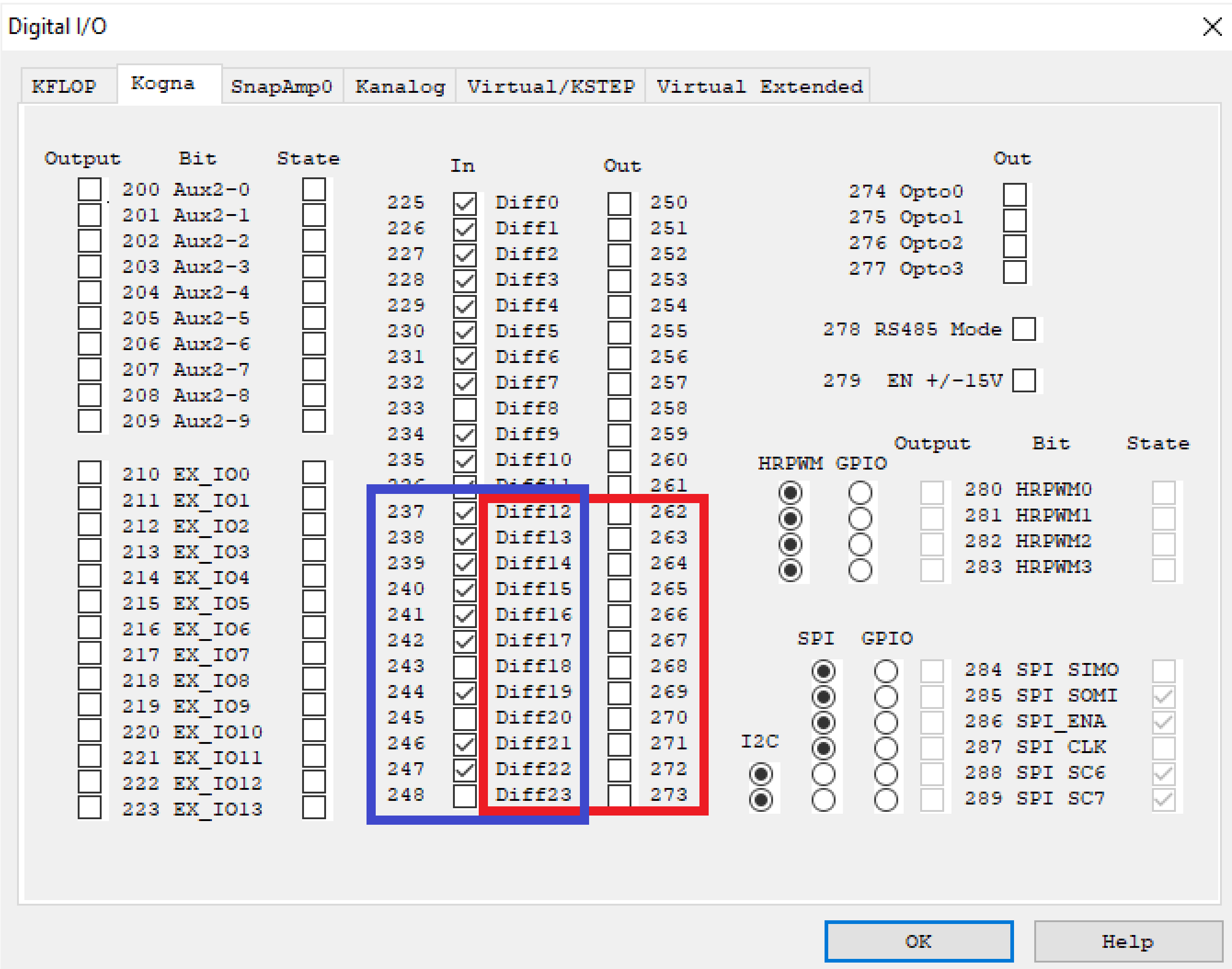
JP14 - Differential Inputs / Differential Outputs



JP14 contains the second half of Kogna's 5V differential outputs (JP10 contains the other half). Enough for 6 Axes of Step/Direction being driven with a Differential Driver. Step/Direction Generators #6 - #11 can be configured to drive pins on JP14. See [Kogna's Step/Direction] for more information. Each Step/Direction Generator has a Step and Direction Output where each Step and Direction output each has a + and - output pin. When the signal is high the + pin is driven near 5V and the - pin is driven near 0V. When the signal is low the + pin is driven near 0V and the - pin is driven near 5V.

To drive single-ended (non-differential) inputs one of the signals may be used with the other left unconnected. In this case the pin is used as a +5V TTL output. To invert the polarity use the - pin rather than the + pin.

If the associated Step/Direction channel is not enabled the differential outputs may be used as 12 General Outputs as IO Bits 262 - 273. The current state can be toggled from the Digital IO Screen as well as the last commanded state displayed as shown in red below:



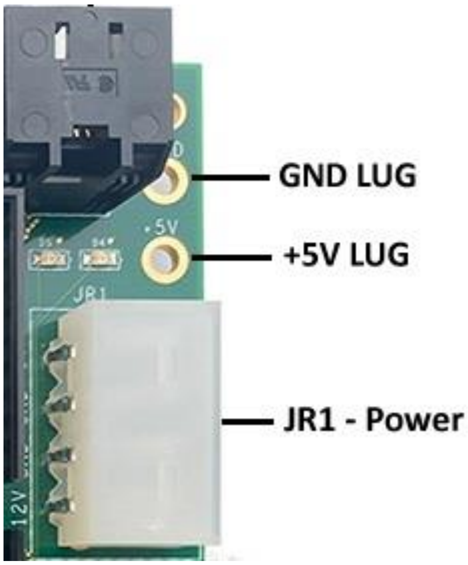
JP14 contains the second half of Kogna's 5V differential inputs (JP8 contains the other half). Enough for 6 A/B quadrature encoders inputs. See [Kogna's Encoders] for more information. Each encoder has an an A and B input where each A and B input has a + and - input pin. When the + pin is at a higher voltage than the - pin then the input is high. When the + pin is at a lower voltage than the - pin the input is low. The + and - pins have a moderate termination resistance of 470 Ohms. Both the + and - pins must be driven for the input to work properly (neither can be left unconnected). The differential inputs may also read as 12 General Inputs as IO Bits 237 - 248 and displayed on the Digital IO Screen as shown in blue above.

JP14 Pin Descriptions

JP14 Pins	Bit	Description
1,2	262	Step #6
3,4	263	Direction #6
5,6	264	Step #7
7,8	265	Direction #7
9,10	266	Step #8
11,12	267	Direction #8
13,14	268	Step #9
15,16	269	Direction #9
17,18	270	Step #10
19,20	271	Direction #10
21,22	272	Step #11
23,24	273	Direction #12
JP14 Pins	Bit	Description
25,26	237	CHA6 - Encoder #14
27,28	238	CHB6 - Encoder #14

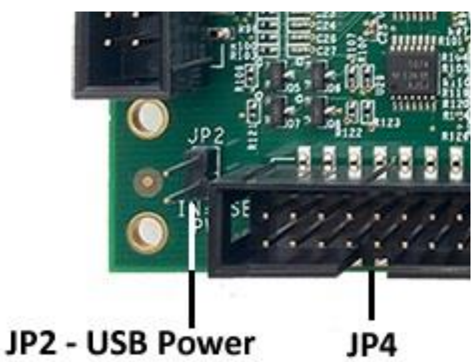
29,30	239	CHA7 - Encoder #15
31,32	240	CHB7 - Encoder #15
33,34	241	CHA8 - Encoder #16
25,36	242	CHB8 - Encoder #16
37,38	243	CHA9 - Encoder #17
39,40	244	CHB9 - Encoder #17
41,42	245	CHA10 - Encoder #18
43,44	246	CHB10 - Encoder #18
45,46	247	CHA11 - Encoder #19
47,48	248	CHB11 - Encoder #19
JP14 Pin	**Supply**	**Description**
49	+5V	+5V Supply
50	0V	Kogna GND

Kogna Power Lugs



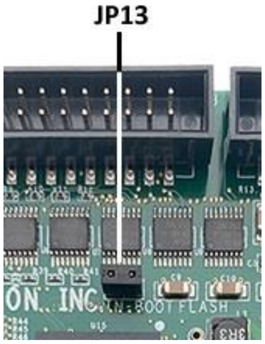
Kogna has 2 Power Lugs GND and +5V that are connected in parallel to GND and +5V on JR1. These can be used to supply (or extract) +5V and GND. They are designed to work with a 4-40 screw/nut. All other Kogna Mounting holes are isolated.

JP2 USB Power Jumper



Jumper JP2 connects USB Power coming in Micro USB Connector J3 to supply KFLOP with +5V. USB Power is limited to 500ma and is sufficient to power Kogna with limited IO loads and the +/-15V Generator turned off. If +/-10V Analog DAC outputs are used the +/-15V Generator must be turned on and more power supplied to Kogna. A +5V @ 2A supply should be sufficient. This is also the case with many outputs driving loads with significant current. For example 50 outputs all driving 10ma loads would increase current by ~500ma. Alternately a USB charger that supplies greater current might be used to power via J3 but obviously the USB COM Port would then not be functional.

JP13 Flash Boot Jumper



Jumper JP3 should normally always be installed to boot from Kogna's Flash memory. Kogna boots in 2 phases. First a primary boot followed by a secondary boot to the latest installed firmware and possibly User Flashed programs. In the case of a corrupted Version of Firmware or bad Flashed User Program Kogna could crash or freeze after booting. After primary booting Kogna delays while flashing its LEDs for approximately 3 seconds. If JP13 is removed during the delay Kogna will remain in the Primary boot loader to allow recovery by Flashing New Version.

Kogna's Encoder Inputs

Kogna contains a total of 20 A/B quadrature encoder inputs normally referenced as channels #0 - 19.

The first 8 operate in the same single-ended LVTTL manner as KFLOP. Where a multiplex bit can be activated to move #0 - 3 input pins from JP7 to JP4. Similarly another multiplex bit can be used to move the #4 - 7 input pins from JP5 to JP6. Kanalog may be added to make these 8 single-ended LVTTL encoder inputs to be 5V differential A/B quadrature encoder inputs

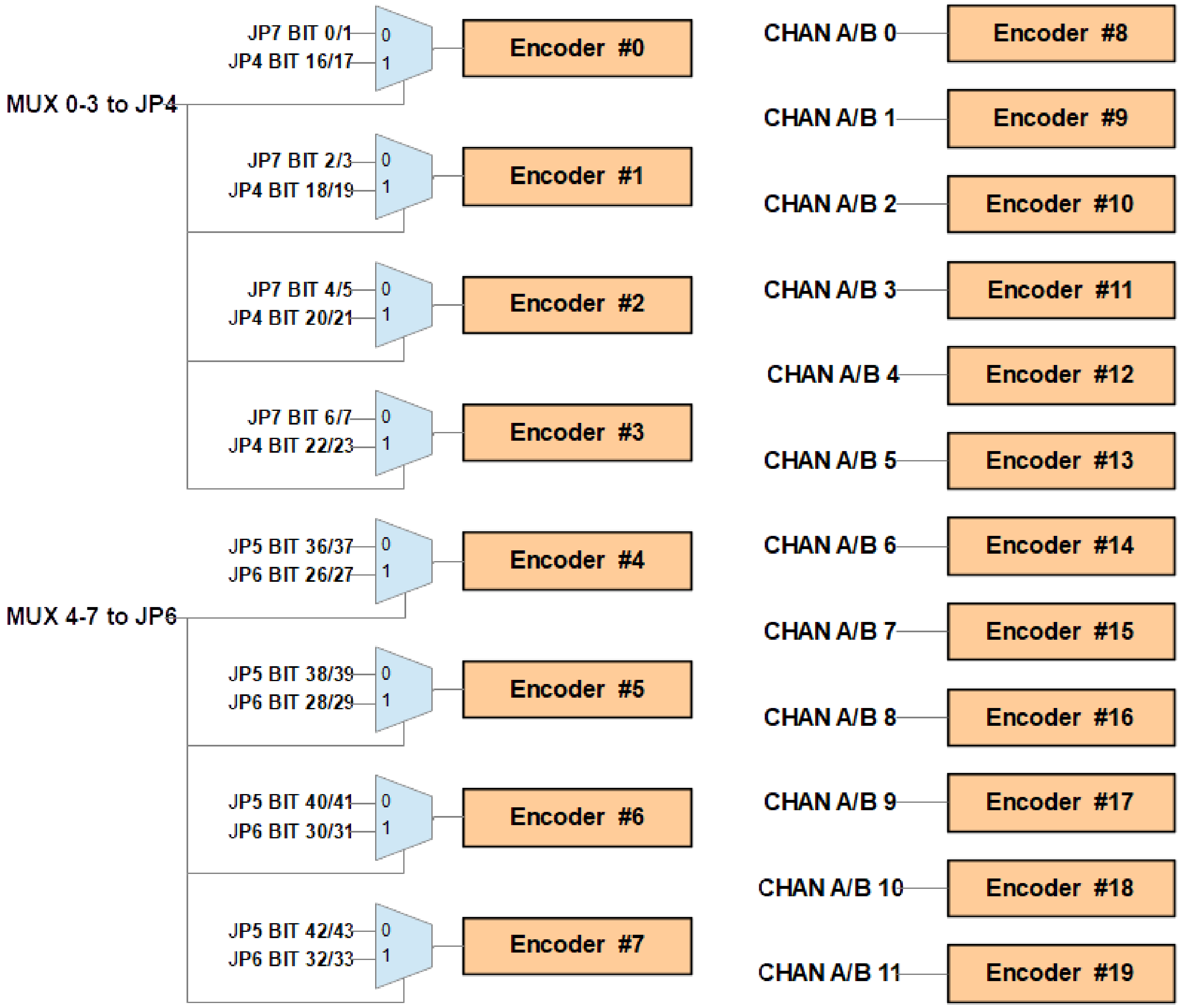
Besides the 8 KFLOP encoder inputs described above Kogna adds 12 5V differential A/B quadrature encoder inputs referenced #8 - 19. These are connected to Kogna's 12 A/B 5V differential receivers. (Encoders #8 - 13) are each connected to the CHA+ CHA- CHB+ CHB- on JP8. (Encoders #14 - 19) are each connected to the CHA+ CHA- CHB+ CHB- on JP14.

In summary:
The first 4 (#0-3) may be connected to JP7 or JP4.
The next 4 (#4-7) may be connected to JP5 or JP6.
The next 12 (#8-20) are connected to 24 (12 pairs) of 5V Differential Receivers.

All IO including Differential Receivers can also be used as GPIO.

Note: Kanalog Option adds Differential Receivers for Encoders #0-7.

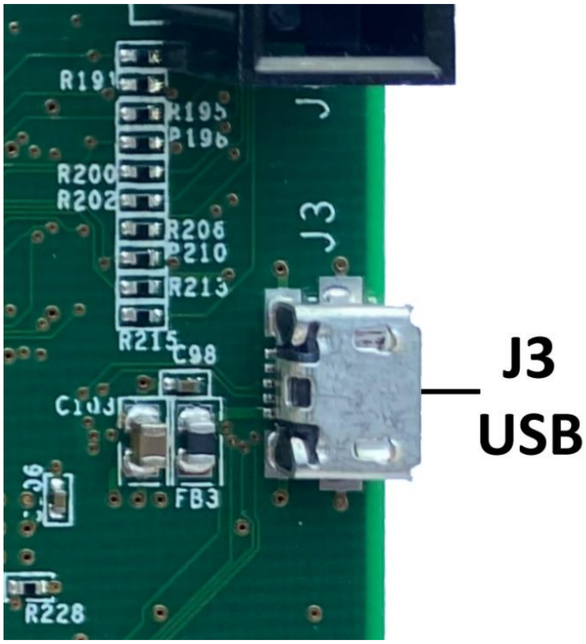
Note: With 8 SnapAmp Encoders being historically referenced as #8-15 when SnapAmps are present, Kogna Encoders #8-15 will be referenced as #16-27.



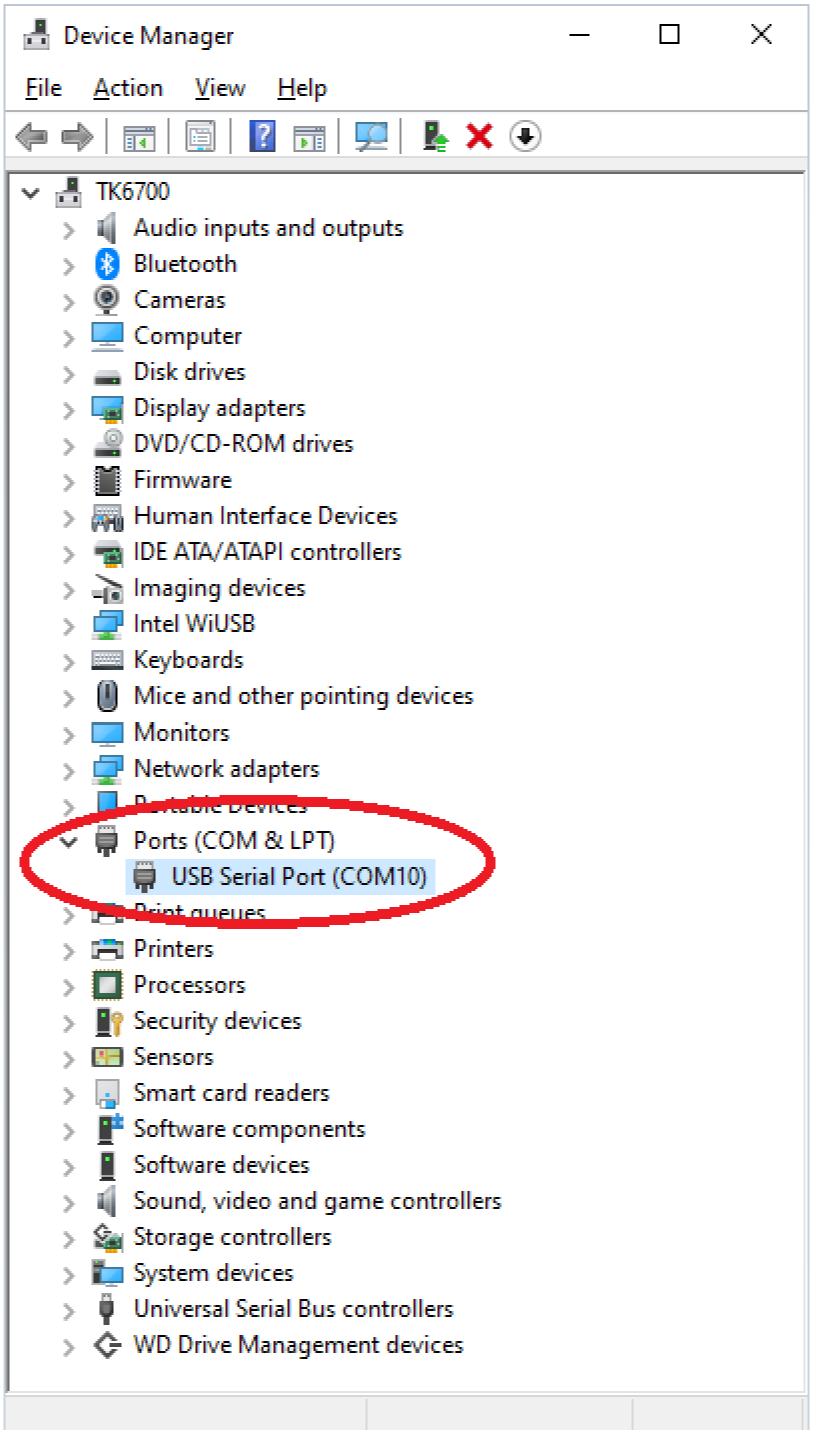
Kogna's 20 Encoder Counters

Kogna's USB UART COM Port

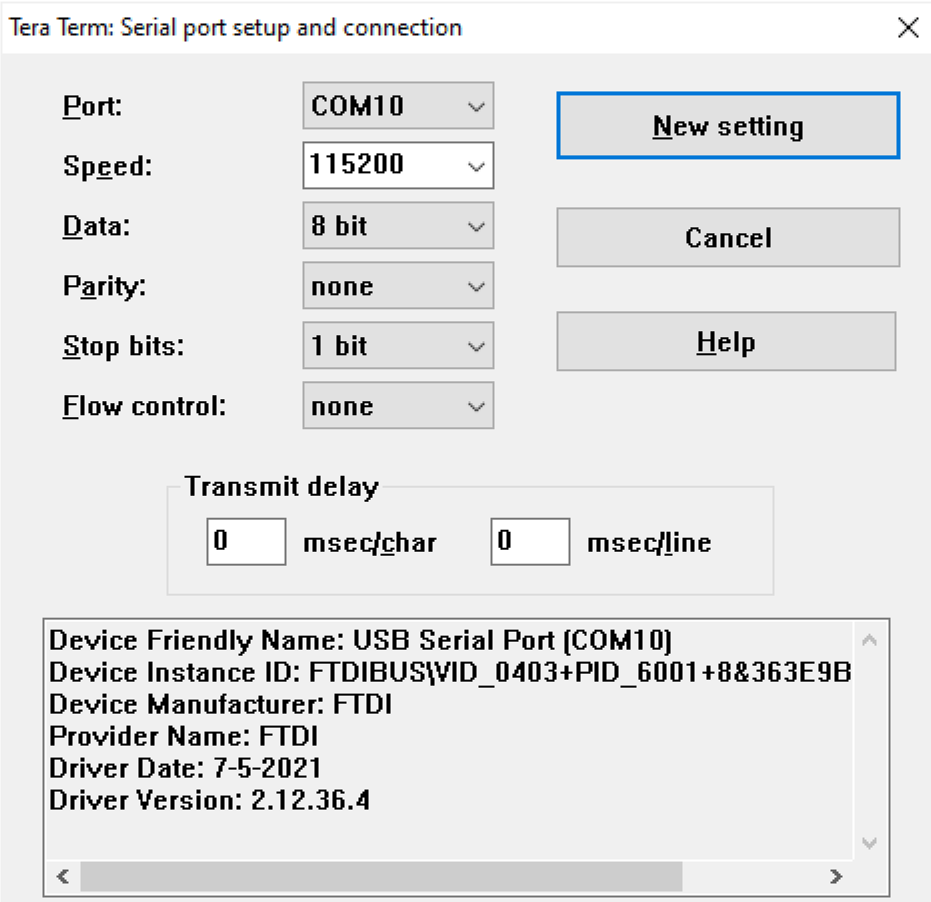
Kogna has a USB Port (J3 on Kogna's backside) that operates as a Virtual COM Port. A micro-USB cable is used to connect between Kogna and the PC Host.



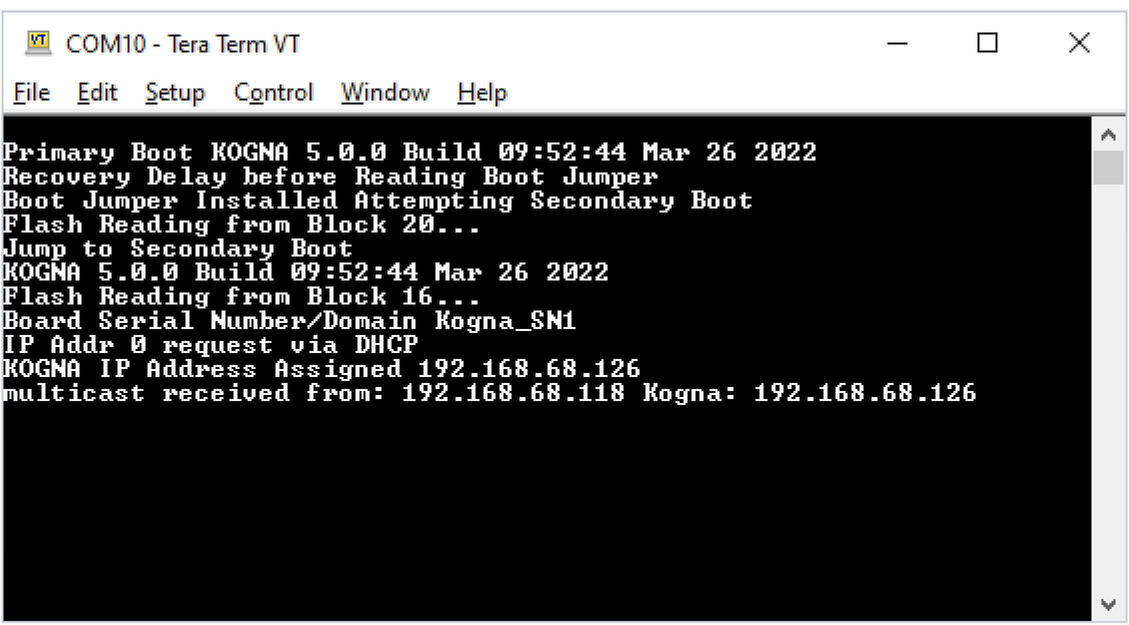
The COM Port can used to display diagnostic information during Boot and about Ethernet activity and errors. Basic commands can be entered to set things like the Ethernet IP Address. When connected a COM port should show up in Windows Device Manager as shown below. Note Windows may assign a different COM Port number than shown below. If your PC has more than one COM Port (or you have more than one Kogna) you might need to try each by trial and error to see which is connected to Kogna. You might also unplug the USB cable to Kogna to see which COM Port is removed to determine which it is. Note the USB COM Port is powered from USB power so should remain connected and powered whether or not Kogna has +5V power applied.



The COM Port is connected to a UART on Kogna's DSP that operates at 115200 Baud. 8-bit data. No Parity. 1 Stop bit. No Flow Control. A Program such as Teraterm or HyperTerminal (or other serial terminal emulators) can be used to display output or enter input commands. TeraTerm can be downloaded from cnet.com.



Below is shown a Primary boot, Delay, then Secondary boot sequence on startup. The Primary boot loader is loaded at the factory. The Secondary boot is of the Firmware Version Flashed into Kogna using Flash New Version. If secondary booting is not successful see here for recovery.



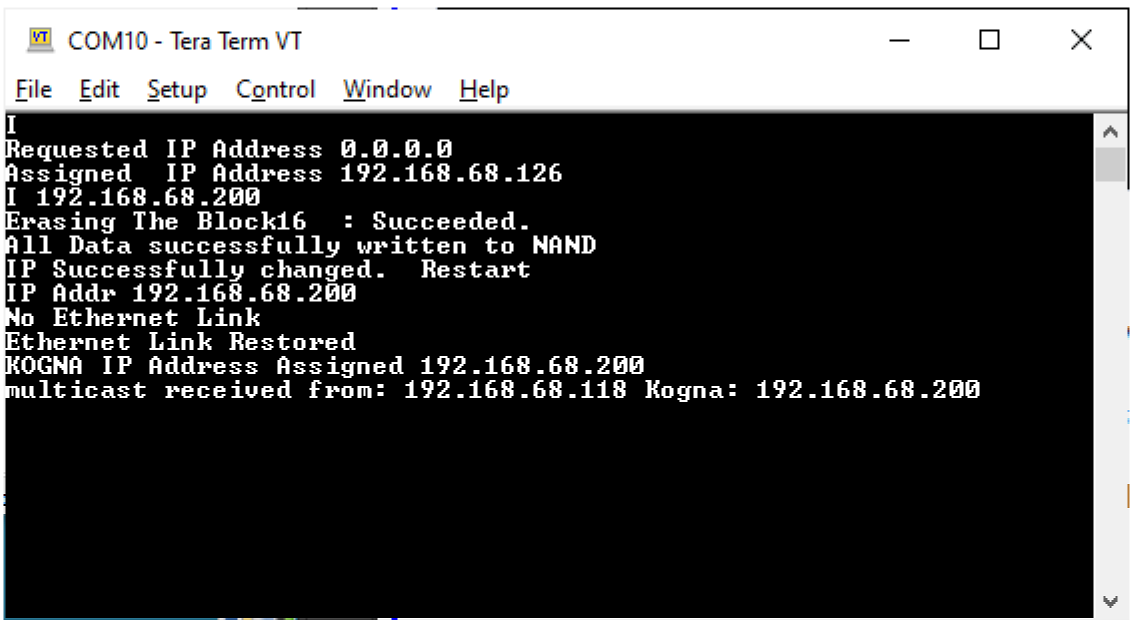
The 'I' command can be used to display or manually set the Ethernet IP Address.

Enter 'I' then Enter to display the requested and assigned IP Address

A requested IP Address of 0 requests the Network DHCP Server to assign the IP Address. An address of 0 may be entered as '0' or '0.0.0.0'.

To manually specify an IP Address enter 4 decimal byte numbers separated by periods. Note changing the IP Address will Flash the setting to Flash memory to be requested on the next Boot or new network connection.

See example below. The 'I' command is used to report that DHCP is used and the Assigned address is 192.168.68.126. Then the IP Address is set to 192.168.68.200. The Ethernet cable is disconnected and re-connect to have the new IP Address assigned. A multicast packet is received from the Host PC 192.168.68.118 and responded to so the PC obtains the new IP Address (and Serial Number).



Configuring Step and Direction Outputs for Kogna

Kogna supports connection to motor amplifiers that utilize step and direction (or quadrature or CW/CCW Pulses) 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. Some Servo Amplifiers may also operate in Step/Direction Mode. 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 RPM / 60 Sec/Min * 200 steps/rev * 128 µsteps/full step = 1.28 MHz µsteps/sec*

Kogna has 16 axes of Step and Direction. Output pulses up to 2.5 MHz can be generated.

Kogna features 12 Axes of +5V Differential Drivers (24 differntial drivers to 48 connector pins). The first 8 Step/Dir Generators can be independently configured to drive a pair of differential Drivers or 3.3V KFLOP compatible pins with an 8-bit R/W FPGA Register called STEP_DIR_TO_DIFF. Each bit muxes the corresponding Step/Dir Generator. Writing a '1' to the bit will Connect the Step/Dir Generator to the differential Driver. Kogna will automatically control the corresponding Differential Enable Bits based on bit 7 of the Axis Configuration OutputChannel number so this is not normally necessary to be set by a User. For example to multiplex all 8 to differential Drivers the following C Code might be used:

```
FPGA(STEP_DIR_TO_DIFF) = 0xFF;
```

The first 8 Axes of Step/Dir outputs can also operate in a Single Ended KFLOP compatible manner where they are normally hard wired to IO bits 8 through 15 on JP7 and IO bits 36 through 43 on JP5. Additionally 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.

Axis output channel settings 0-255 are allowed for Kogna. Although only 16 Step and Direction generators are available for Kogna. By adding specific constants to the channel number the operation mode and output mode of each Step/Dir Generator can be changed independently. Not all Output Drive modes are valid for all Generators.

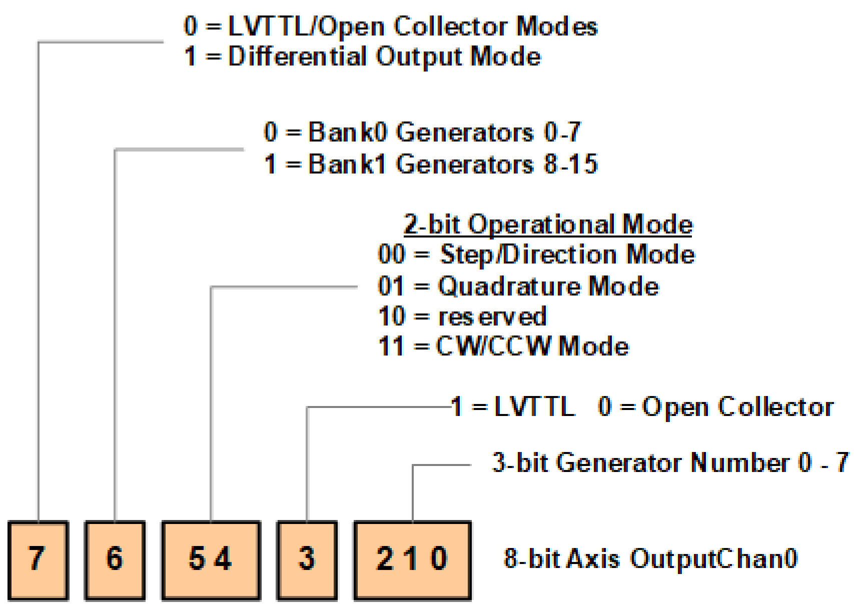
Adding 8 to the channel number uses the same generator to be used except the output pins are actively driven (high and low) as 3.3V LVTTL signals instead of only driven low as open collector outputs.

Adding 16 to the channel number switches the generator from Step/Dir to Quadrature output mode.

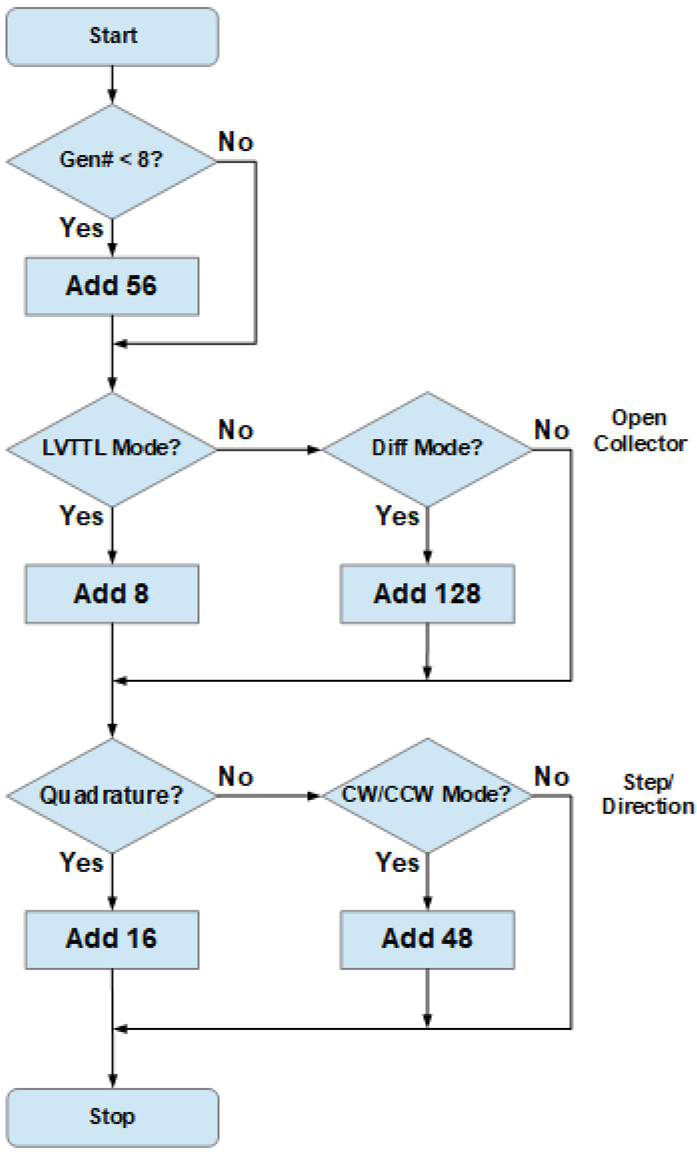
Adding 48 to the channel number switches the generator from Step/Dir to CW/CCW output mode.

Adding 64 to the channel number switches to the second bank of 8 generators (this is to maintain legacy compatibility with KFLOP's channel 8 Axes numbering scheme)

Adding 128 to the channel number selects Differntial Output mode and pins (only available on Generators 0-11)



Step/Direction Generator and Modes Key
Note not all mode combinations are available for all Generators



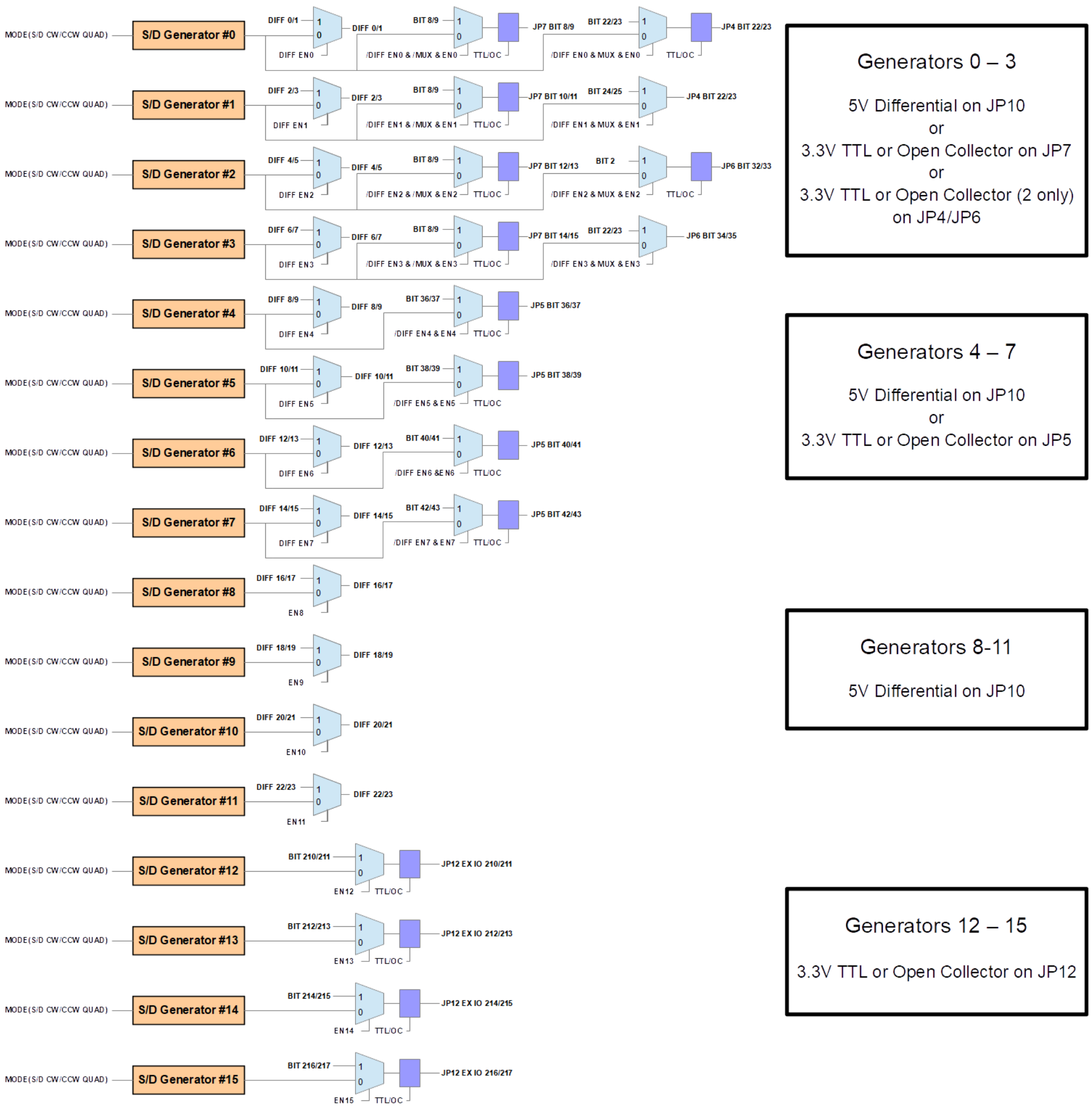
Step/Direction OutputChan0 Configuration Flow Chart
Note not all mode combinations are available for all Generators

Step/Dir Generators 8-11 will always drive diffential Drivers 16-23 whenever they are enabled.

Any IO pin not being driven by a Step/Dir Generator may be used for other purposes such as GPIO.

If an axis channel is selected as a Step and Direction axis and enabled, the corresponding 2 output pins will be automatically configured as outputs (if necessary) and they may not be used as general purpose IO.

See diagram below. For a larger image click here.



Kogna's 16 Step/Direction Generator Options

First 12 may be connected to 24 (12 pairs) of 5V Differential Drivers
First 8 may be connected to 3.3V TTL/Open Collector IO instead (KFLOP Compatibility)
Remaining 4 are 3.3V TTL/ Open Collector outputs
Any IO including Differential Drivers not driven by Step/Dir Generator may be used as GPIO
Each Step/Dir Generator may independently operate in Step/Dir, or CW/CCW, or AB Quadrature Modes.

The tables below lists the IO Bit and Pin numbers associated with the Step/Dir Generator and the Mode and Multiplex option:

Generators 0-3 have 3 options:

	Enable Diff = 1		Enable Diff = 0			
	Mux = X		Mux = 0		Mux = 1	
Signal	**IO Bit**	**Pins**	**IO Bit**	**Pin**	**IO Bit**	**Pin**
Step 0	250	JP10 - 1 & 2	8	JP7 - 15	22	JP4 - 13
Direction 0	251	JP10 - 3 & 4	9	JP7 - 16	23	JP4 - 14
Step 1	252	JP10 - 5 & 6	10	JP7 - 17	24	JP4 - 15
Direction 1	253	JP10 - 7 & 8	11	JP7 - 18	25	JP4 - 16
Step 2	254	JP10 - 9 & 10	12	JP7 - 19	32	JP6 - 13
Direction 2	255	JP10 - 11 & 12	13	JP7 - 25	33	JP6 - 14
Step 3	256	JP10 - 13 & 14	14	JP7 - 21	34	JP6 - 15
Direction 3	257	JP10 - 15 & 16	15	JP7 - 22	35	JP6 - 16

Generators 4-7 have 2 options:

	Enable Diff = 1		Enable Diff = 0	
Signal	IO Bit	Pins	IO Bit	Pin
Step 4	258	JP10 - 17 & 18	36	JP5 - 1
Direction 4	259	JP10 -19 & 20	37	JP5 - 2
Step 5	260	JP10 - 21 & 22	38	JP5 - 3
Direction 5	261	JP10 - 23 * 24	39	JP5 - 4
Step 6	262	JP10 - 25 & 26	40	JP5 - 5
Direction 6	263	JP10 - 27 & 28	41	JP5 - 6
Step 7	264	JP10 - 29 & 30	42	JP5 - 7
Direction 7	265	JP10 - 10 & 32	43	JP5 - 8

Generators 8-11 are Differntial only:

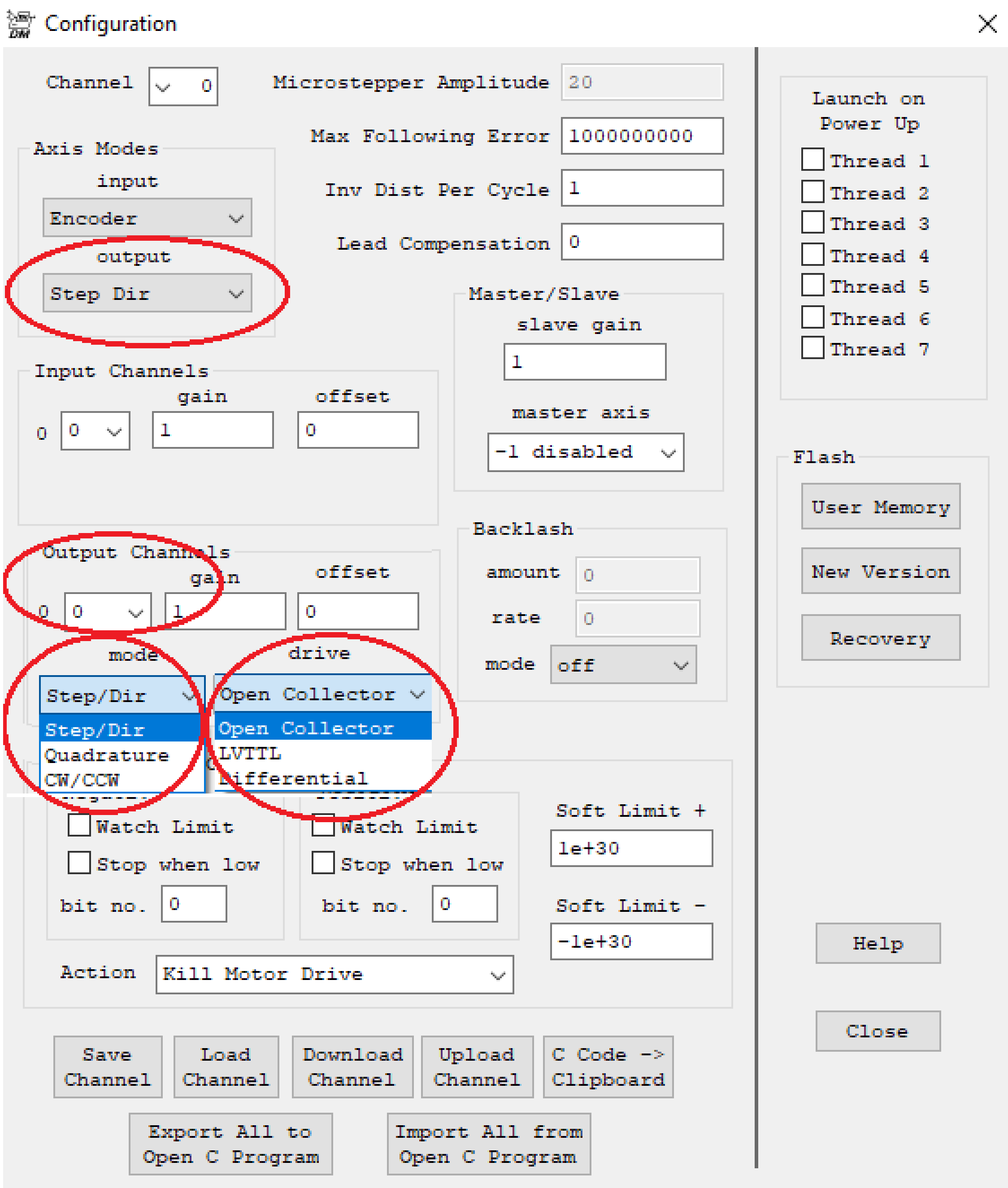
Signal	IO Bit	Pins
Step 8	266	JP14 - 1 & 2
Direction 8	267	JP14 - 3 & 4
Step 9	268	JP14 - 5 & 6

Direction 9	269	JP14 - 7 & 8
Step 10	270	JP14 - 9 & 10
Direction 10	271	JP14 - 11 & 12
Step 11	272	JP14 - 13 & 14
Direction 11	273	JP14 - 15 & 16

Generators 11-15 3.3V IO only:

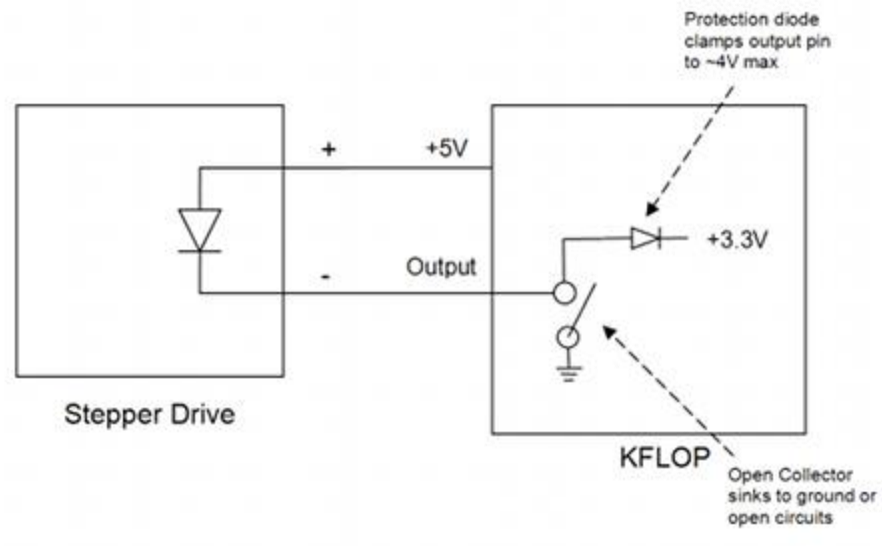
Signal	IO Bit	Pin
Step 12	200	JP12 - 11
Direction 12	201	JP11 -12
Step 13	202	JP12 - 13
Direction 13	203	JP12 - 14
Step 14	204	JP12 - 15
Direction 14	205	JP12 - 16
Step 15	206	JP12 - 17
Direction 15	207	JP12 - 18

The simplest method to configure a Step/Dir Axis and its various channels and modes is to use the KMotion.exe Configuration Screen shown below. To configure an axis as step and direction, on the Configuration Screen select output mode as "Step Dir" and select which of the 16 available Step and Direction generators should be used by setting the appropriate Output Channel 0 setting. Also select the appropriate operational mode and output Drive mode. See below.



If your amplifier has opto coupler inputs driven by +5V then a 5V Differential Drive Output is the best choice. Either as single ended 5V signals (by using one of the 5V differential outputs) or differential 5V signals. Differential signals allow faster step rates.

If a Differntial Drive Output is not available then a 3.3V output must be used. In this case 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.

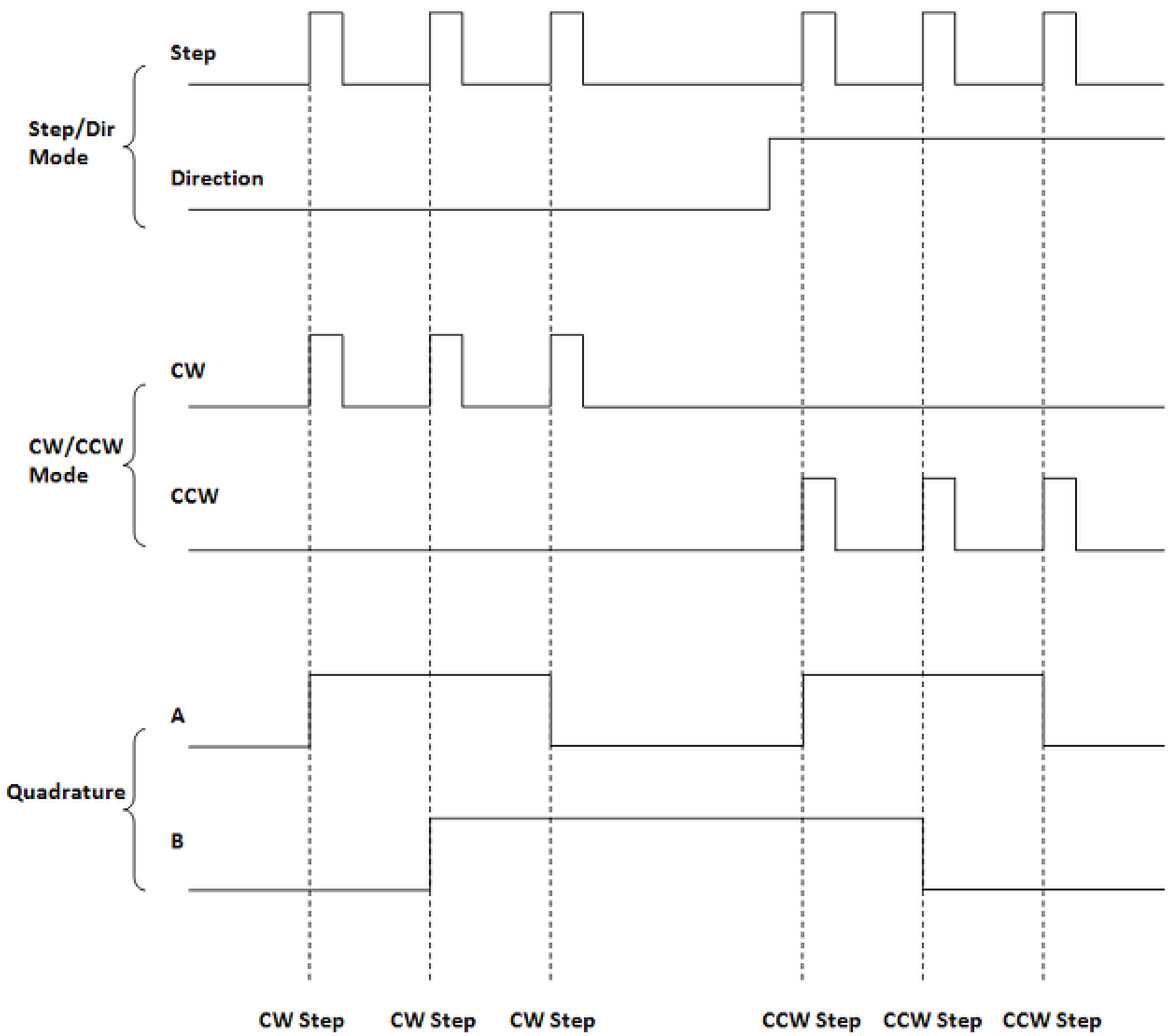


However if the amplifier has standard logic inputs then LVTTL outputs should work better as the pin is actively driven high and low.

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 and any Direction Setup Time and Minimum pulse time is avoided. Quadrature mode is selected by adding 16 to the Output Channel Number.

CW/CCW Mode is also available. In this mode instead of having a Direction signal there is a separate output for CW and CCW Pulses. This mode avoids any Direction Setup Time as well. CW/CCW mode is selected by adding 48 to the Output Channel Number.

Example 3 CW Steps followed by 3 CCW Steps in All 3 Possible Modes



The table below summarizes the Output Channel Number to specify to select the desired Generator and Mode:

	Step/Dir mode +0			Quadrature mode +16			CW/CCW mode +48		
Generator	Differential +128	Open Collector +0	LVTTL +8	Differential +128	Open Collector +0	LVTTL +8	Differential +128	Open Collector +0	LVTTL +8
0	128	0	8	144	16	24	176	48	56
1	129	1	9	145	17	25	177	49	57
2	130	2	10	146	18	26	178	50	58
3	131	3	11	147	19	27	179	51	59
4	132	4	12	148	20	28	180	52	60
5	133	5	13	149	21	29	181	53	61
6	134	6	14	150	22	30	182	54	62
7	135	7	15	151	23	31	183	55	63
8	192	64	72	208	80	88	192	112	120
9	193	65	73	209	81	89	193	113	121
10	194	66	74	210	82	90	194	114	122
11	195	67	75	211	83	91	195	115	123
12	N/A	68	76	N/A	84	92	N/A	116	124
13	N/A	69	77	N/A	85	93	N/A	117	125
14	N/A	71	78	N/A	86	94	N/A	118	126
15	N/A	71	79	N/A	87	95	N/A	119	127

For a table mapping from Output Channel Number to Generator and mode see here.

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 steps/sec, steps/sec^2, and steps/sec^3 respectively. From Kogna's perspective a "step" is the movement for a step pulse to the Drive which may or may not actually be a micro-step depending on the Drive and its settings.

Global Register sets Pulse Width, Polarity, Multiplexor

To change the Step/Dir Pulse width, Step Pulse Polarity, and connector multiplexor for channels 0-3 a programmable register in Kogna's FPGA may be used.

Kogna 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 x 2us) = 250KHz.

Kogna sets the Direction output 1.92us before generating a Step Pulse. With the maximum Step Pulse length of 3.78us the maximum Direction Setup time to the trailing edge of the Step Pulse is 5.7us.

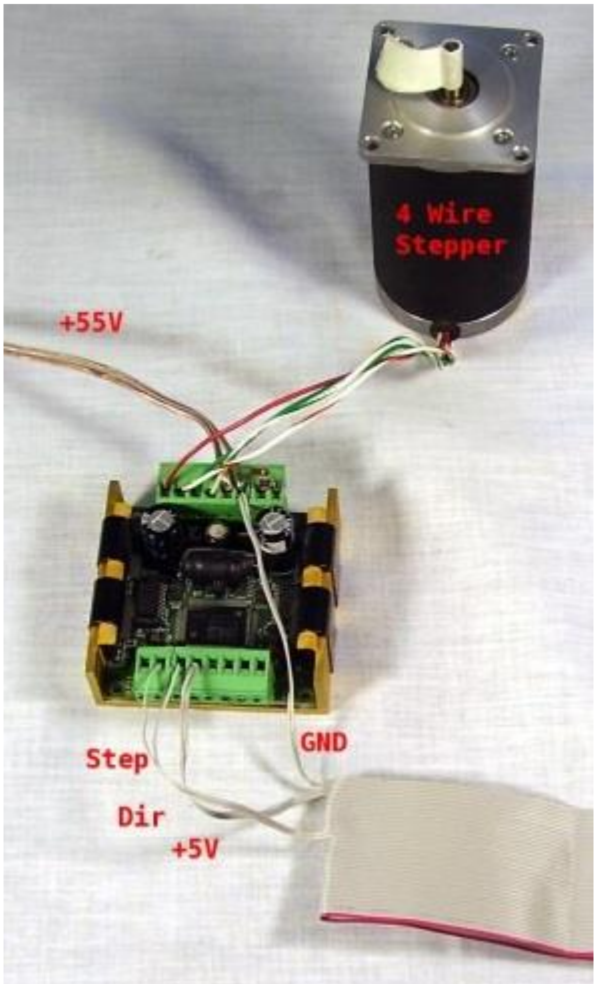
Setting Bit-6 high of the register can be set high to multiplex Step/Dir generators 0-3 from JP7 to JP4 and JP6.

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.

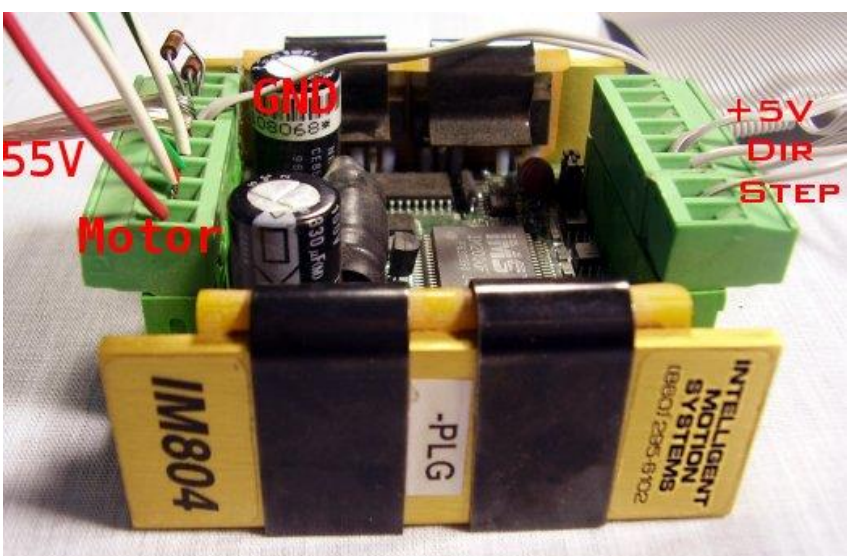
A User C Program must be used to change the FPGA register. The following statement should be used:

```
        FPGA(STEP_PULSE_LENGTH_ADD)=32;                                       // set the pulse time to ~ 2µs
        FPGA(STEP_PULSE_LENGTH_ADD)=32 + 0x40;                // set the pulse time to ~ 2µs and multiplex to JP4 and JP6
        FPGA(STEP_PULSE_LENGTH_ADD)=32 + 0x80;                // set the pulse time to ~ 2µs and pulse the Step High
        FPGA(STEP_PULSE_LENGTH_ADD)=32 + 0x40 + 0x80  // set the pulse time to ~ 2µs, mux to JP4 and JP6, and pulse the Step High
```

Example Configuration for a IM804 Amplifier and Rapidsyn Stepper

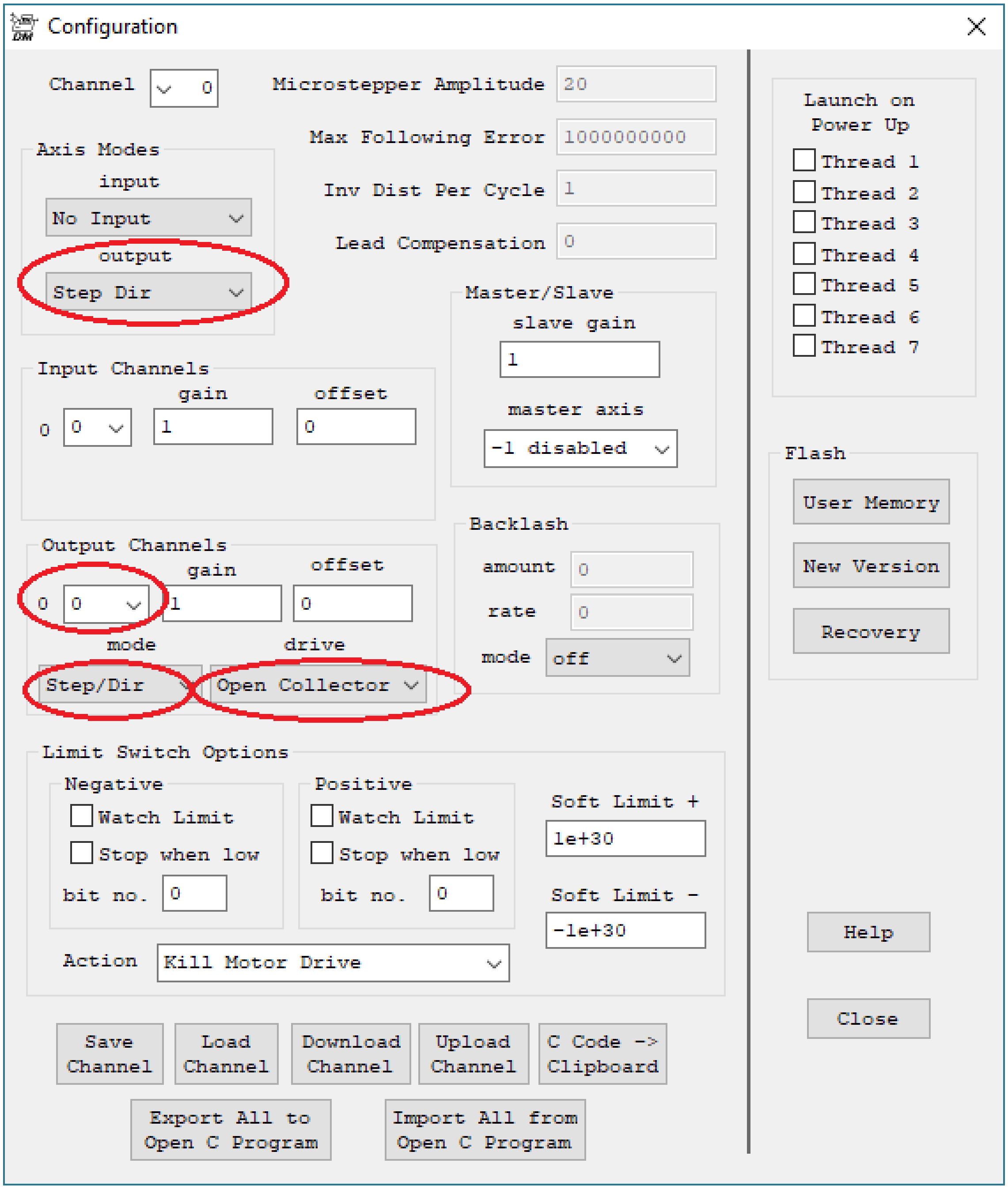


Layout - KMotion - Amplifier - Motor

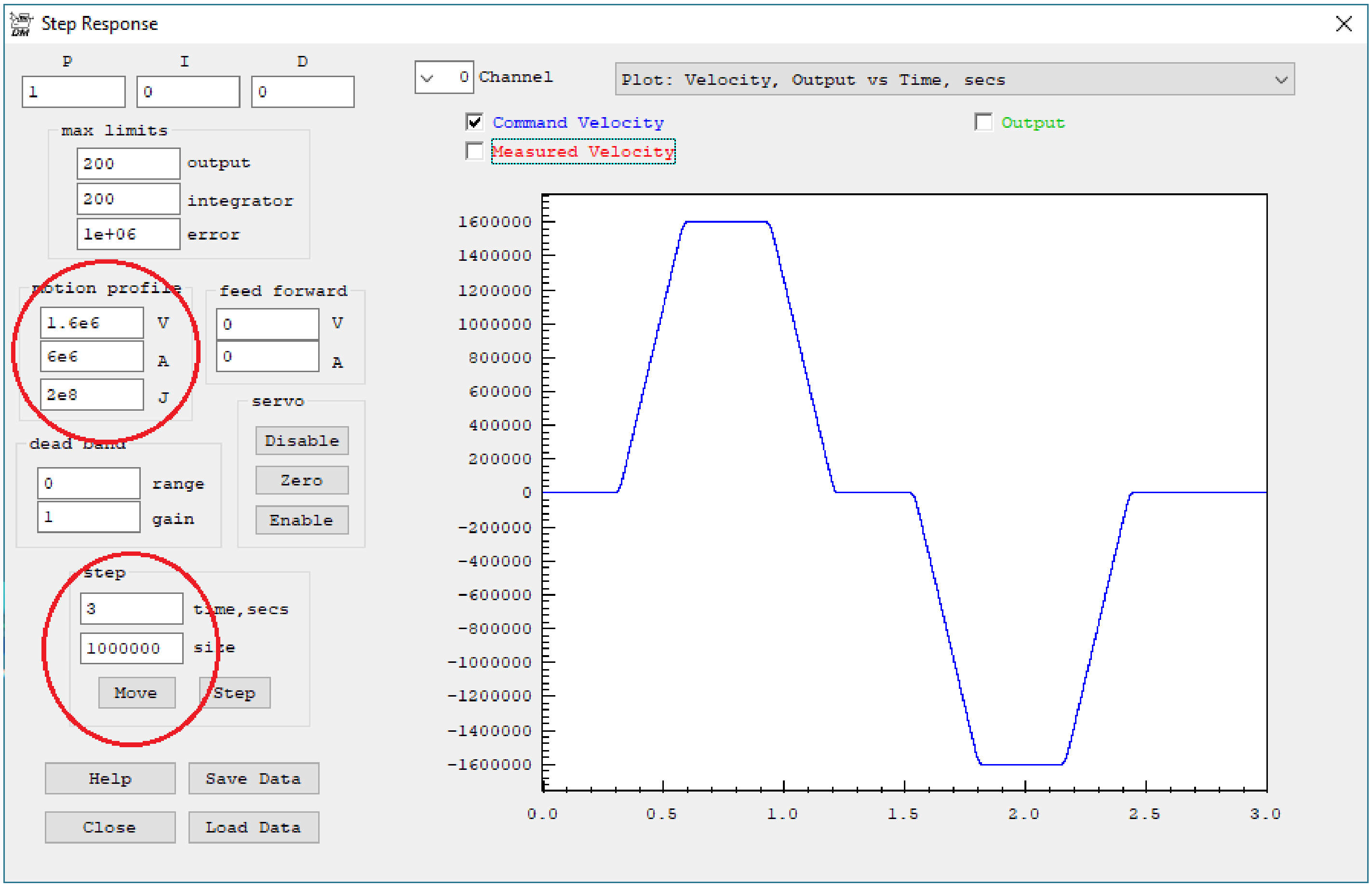


Amplifier Connections - resistors set coil currents (1500 ohms = 3A) and idle standby current (270 ohms = 0.5A). DIP Switches set to ON-OFF-OFF-ON for 128:1 micro-stepping.

Rapidsyn stepper motor. Note coil winding's centertaps are not used.



Axis channel 0 configures as "Step Dir" mode and using Step and Direction generator 0.



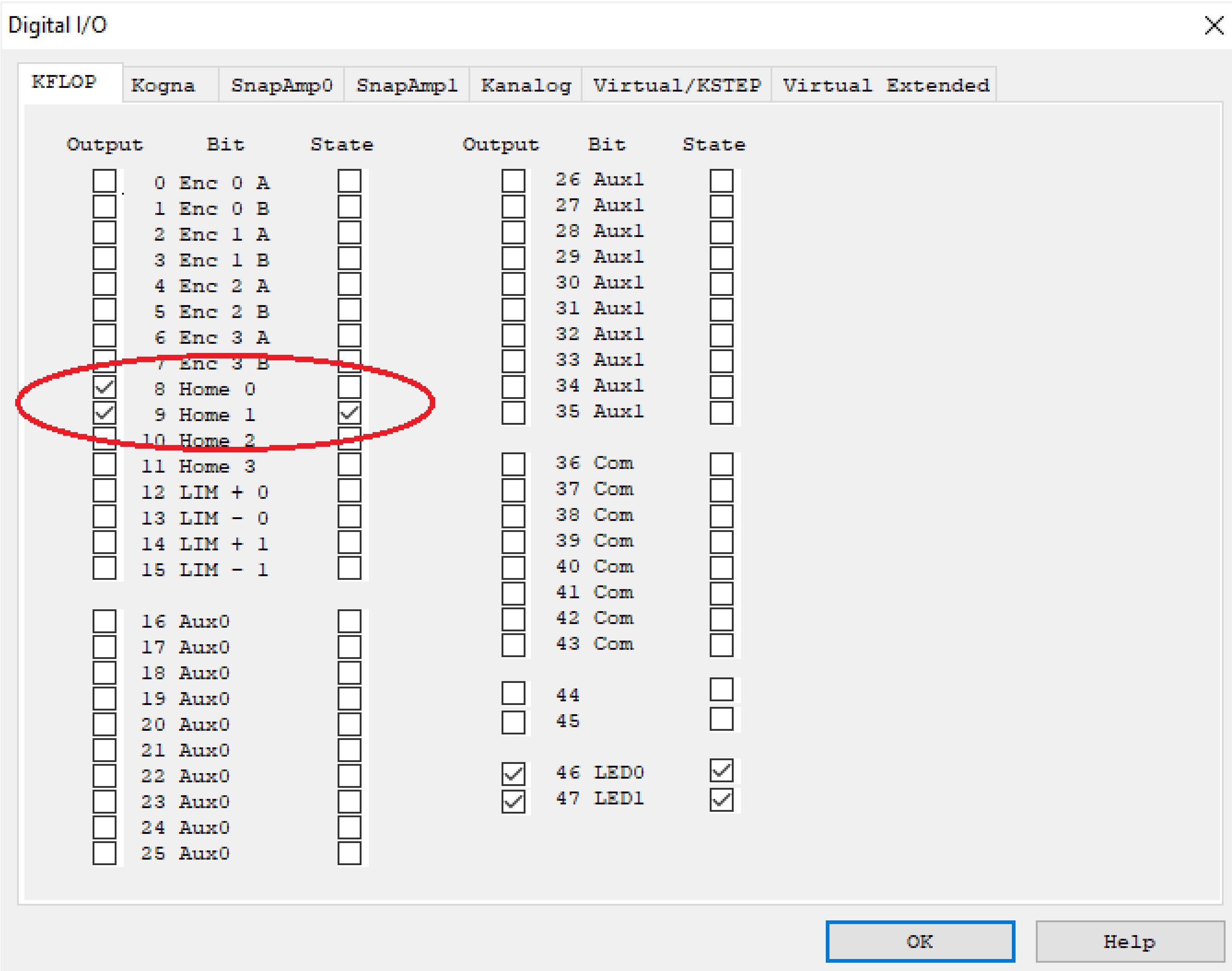
Max Velocity 1.6 x 10^6 steps/sec (1.6e6 / 128 = 12,500 sps = 3750 RPM). Note this (and most) steppers lose most all of their useful torque above several thousand full steps per second. The high speed demonstrated here is intended to demonstrate KMotion's ability to generate high pulse rates.

Max Acceleration 6 x 10^6 steps/sec^2.

Max Jerk 2 x 10^8 steps/sec^3.

1 x 10^6 count Move performed and data captured over 3 seconds.

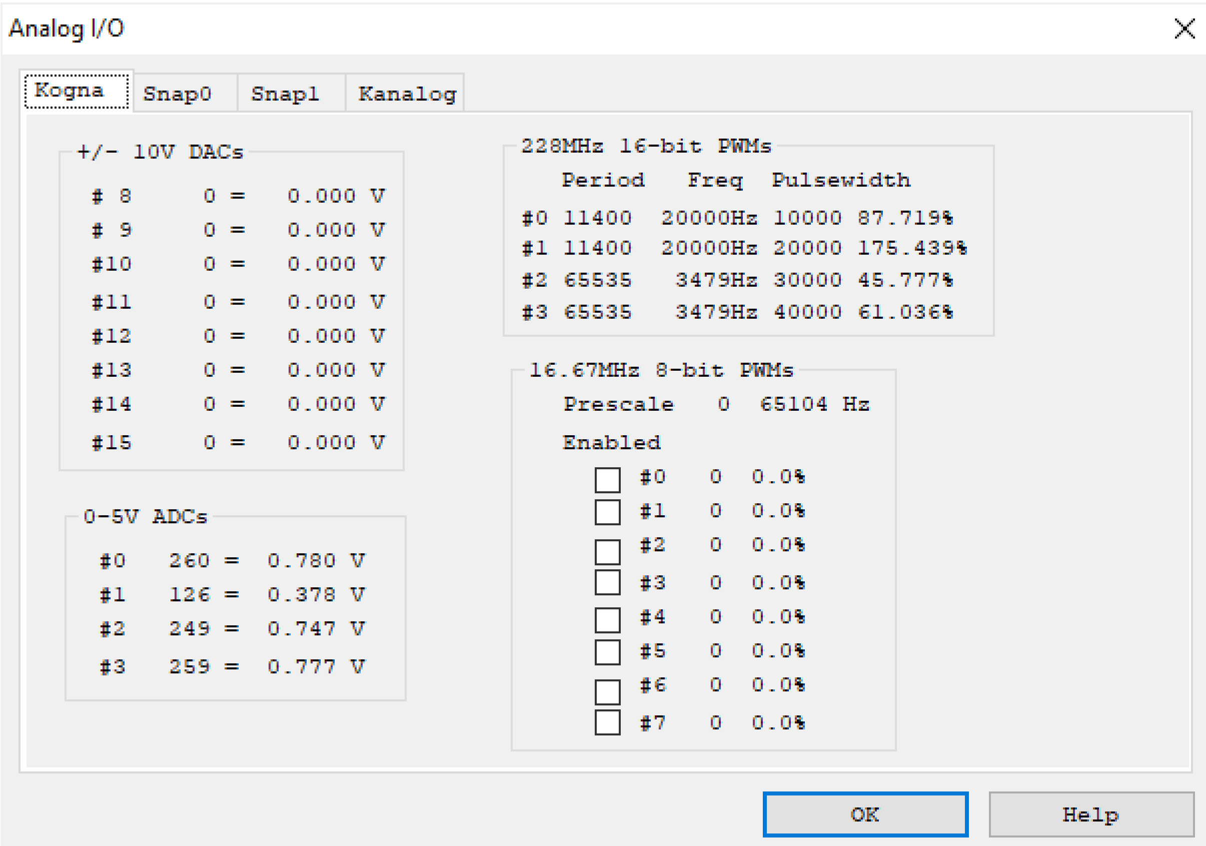
Pushing Move downloads all parameters, enables the axis, performs the movement, and plots the result.

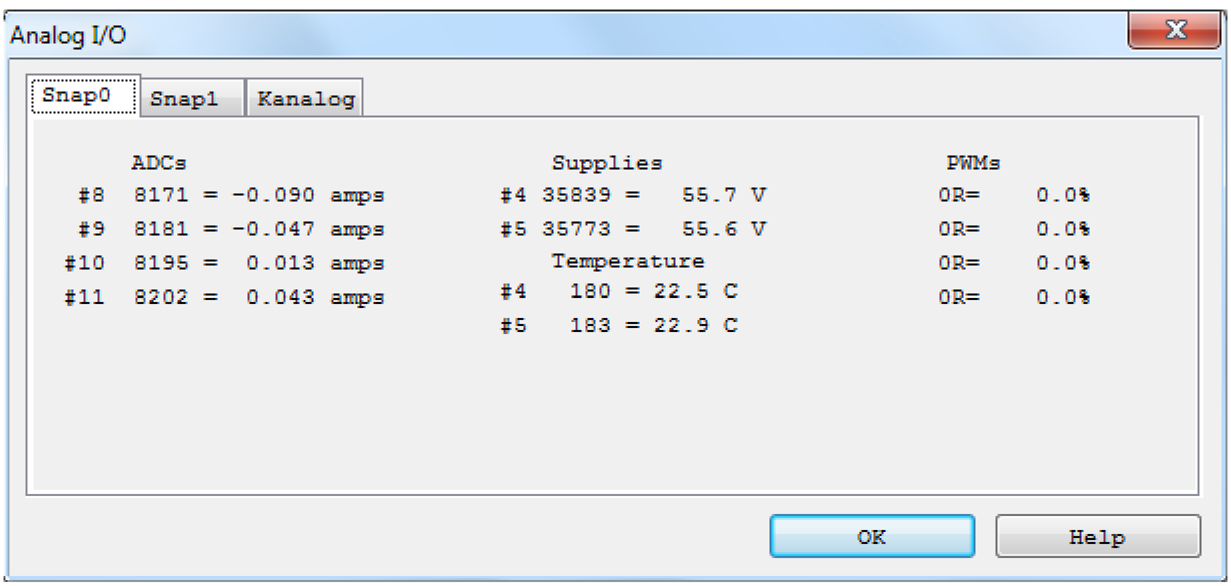


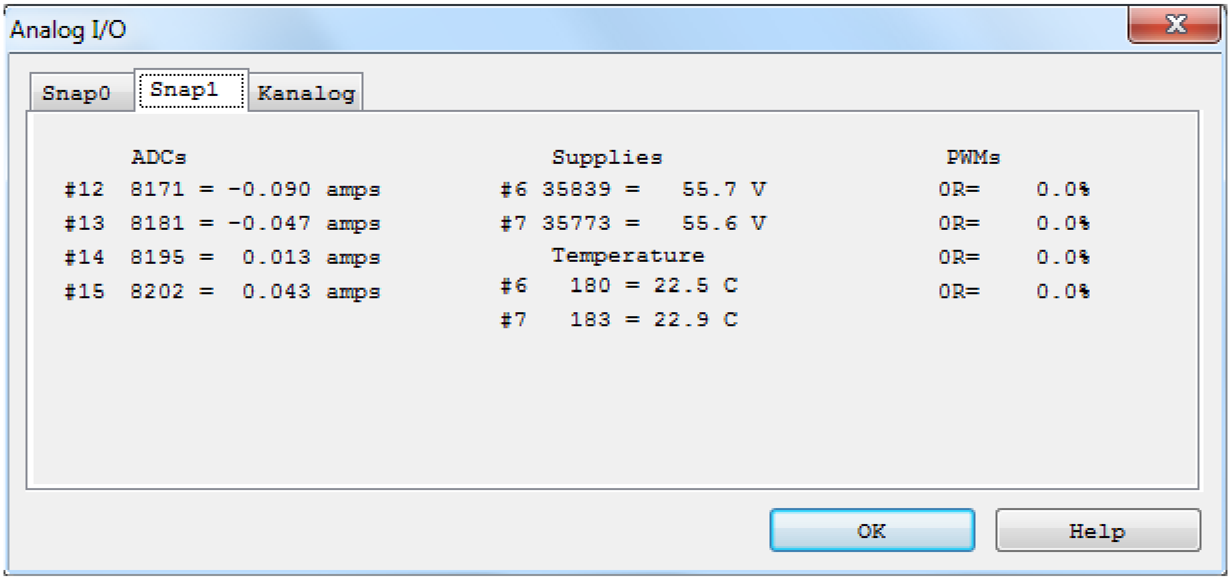
Note after the axis has been enabled, digital IO bits 8 and 9 have automatically been configured as outputs for use by Step and Direction Generator #0.

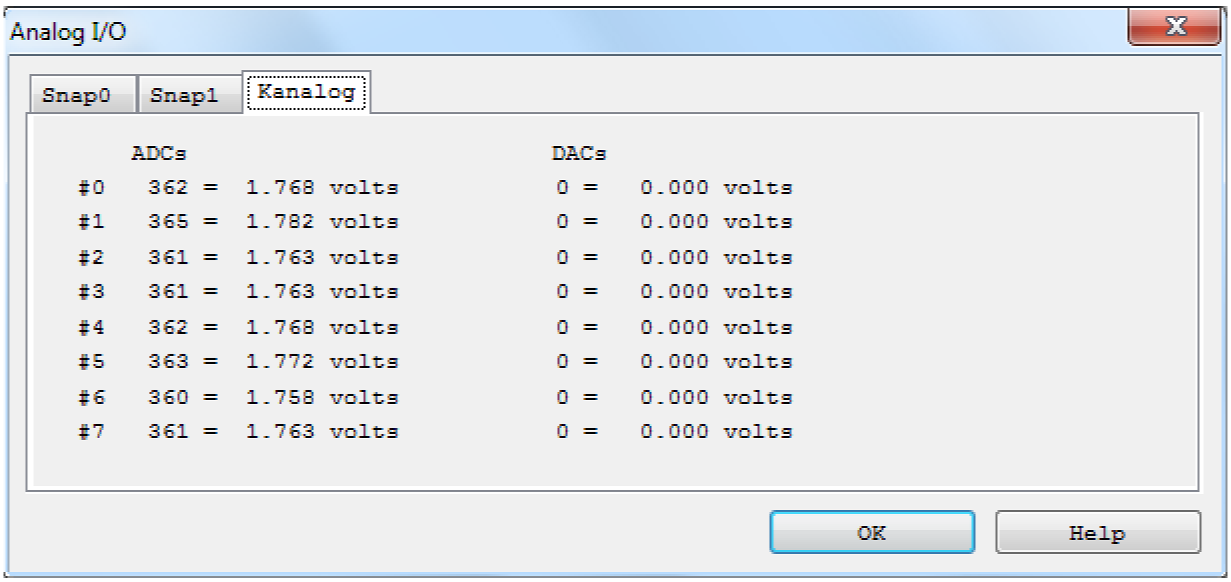
Watch a video showing slow and smooth acceleration of this configuration up to 12,500 full steps per second (1.6 MHz @ 128 microsteps/full step).

Kogna – Analog I/O Status Screen









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

- Kogna DAC Settings
- Kogna ADC Readings
- Kogna Hi-Res PWM Settings
- Kogna PWM Settings
- 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

See links below for:

Kogna's DACs

Kogna's ADCs

Kogna's 16-bit Hi-Res PWMs

Kogna's 8-bit PWMs

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.

PWMs

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./p>

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.

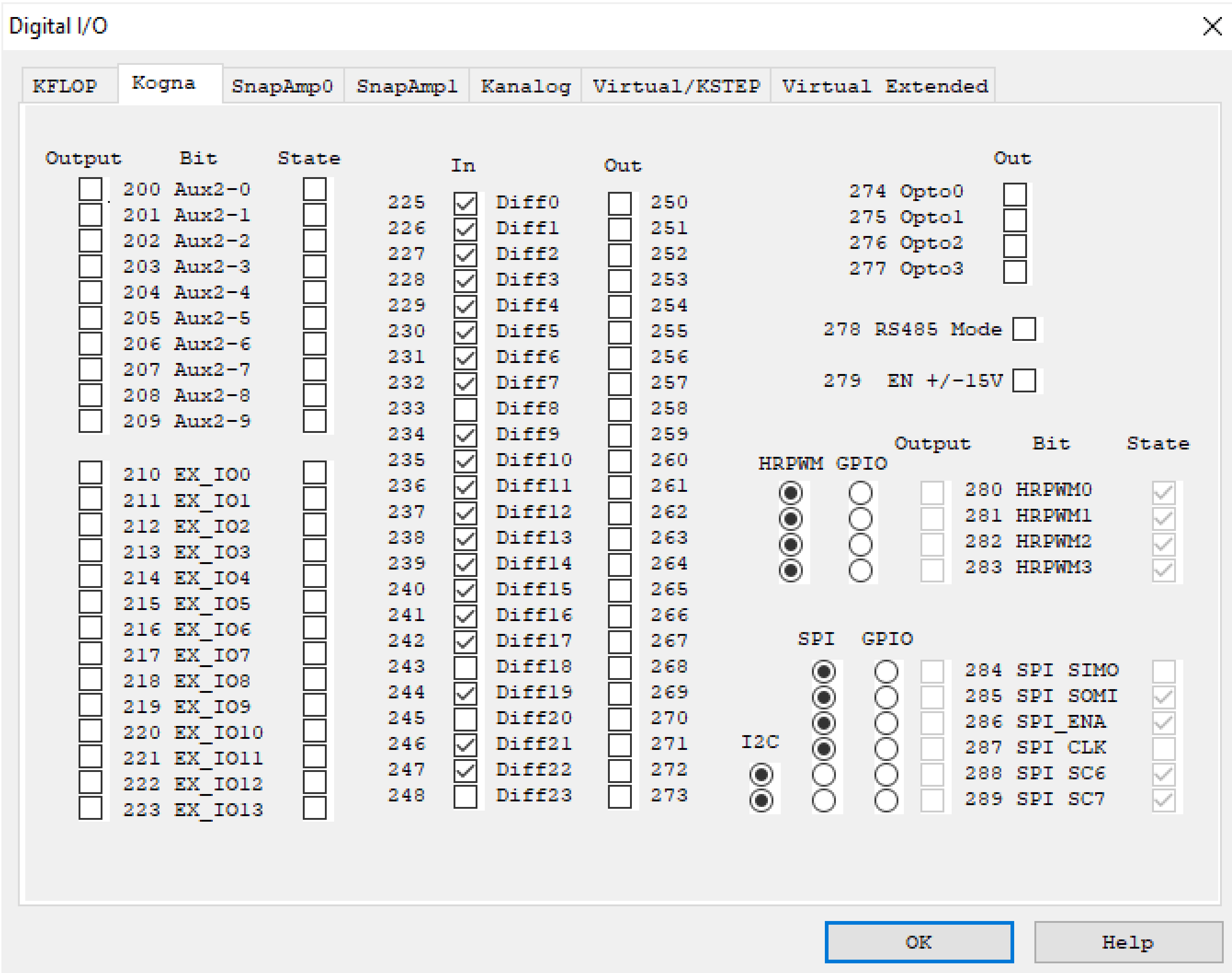
A PWM channel may also operate in current loop mode. In current loop mode the PWM duty cycle is automatically adjusted in order to maintain the commanded current. The PWM channel status will show the command value in counts (-4095 ... +4095), followed by a "C", and the percent of full scale.

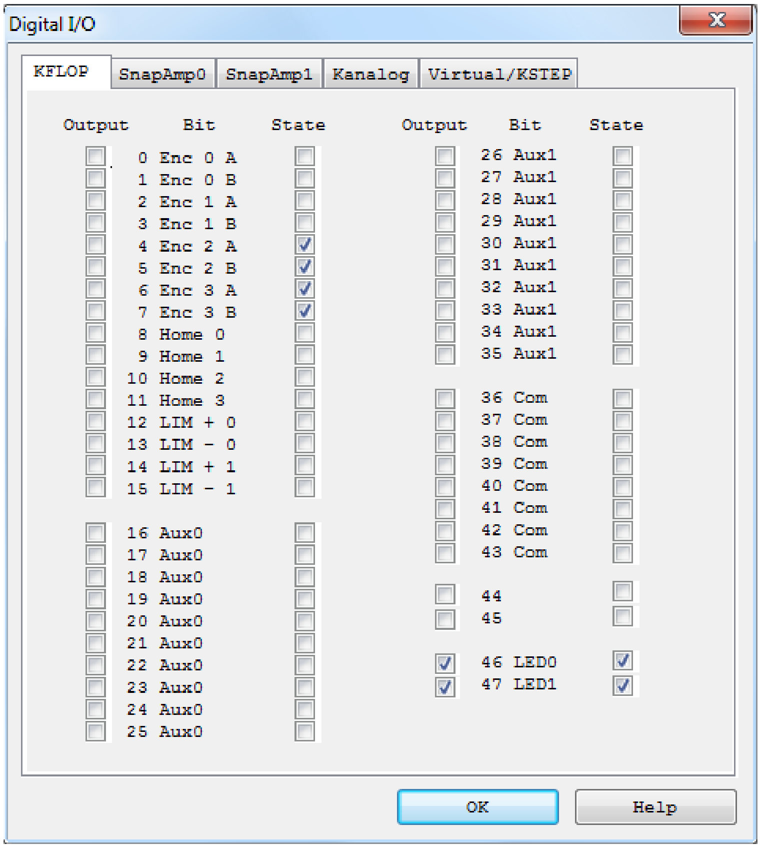
If a pair of PWM channels is operating in 3 Phase mode, the PWM channel status will show the value in counts (-230 ... 230) after the first PWM channel and the phase angle in degrees after second PWM channel.

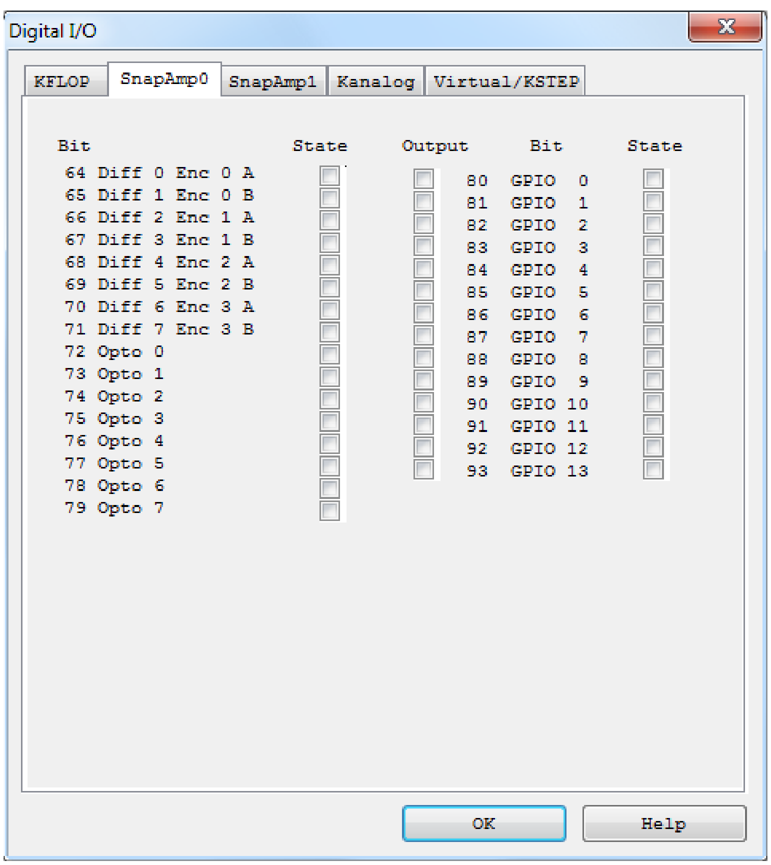
The example status below shows PWM channels 8 and 9 operating in 3 phase mode, PWM channel 10 operating in Normal mode, and PWM channels 11 operating in Recirculating mode.

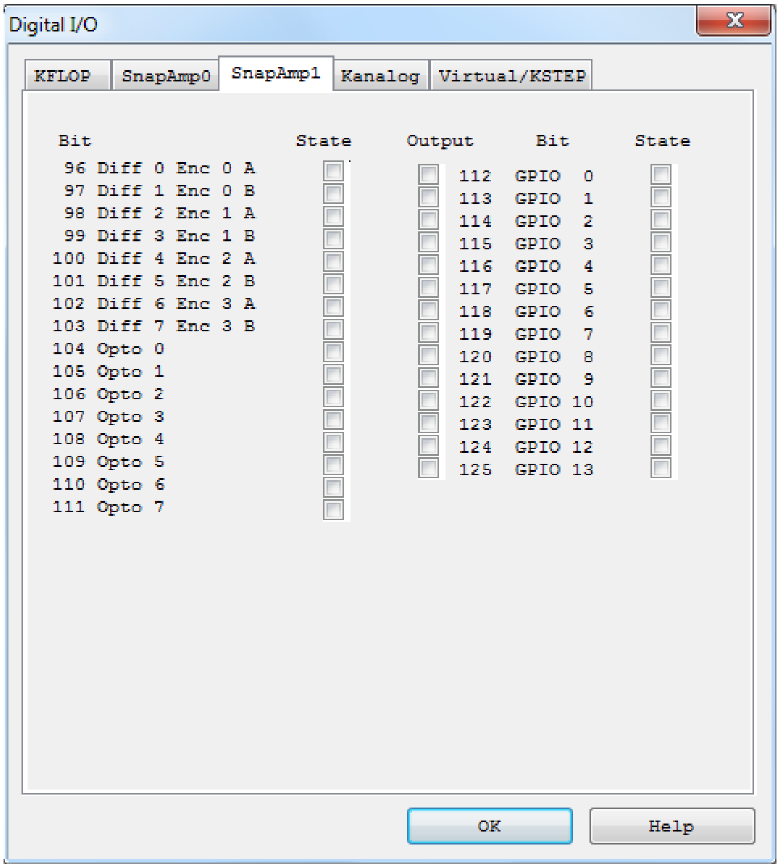
```
     PWMs
 100 =   43.5%
   120 degrees
  50 =   19.5%
 200R=   39.1%
```

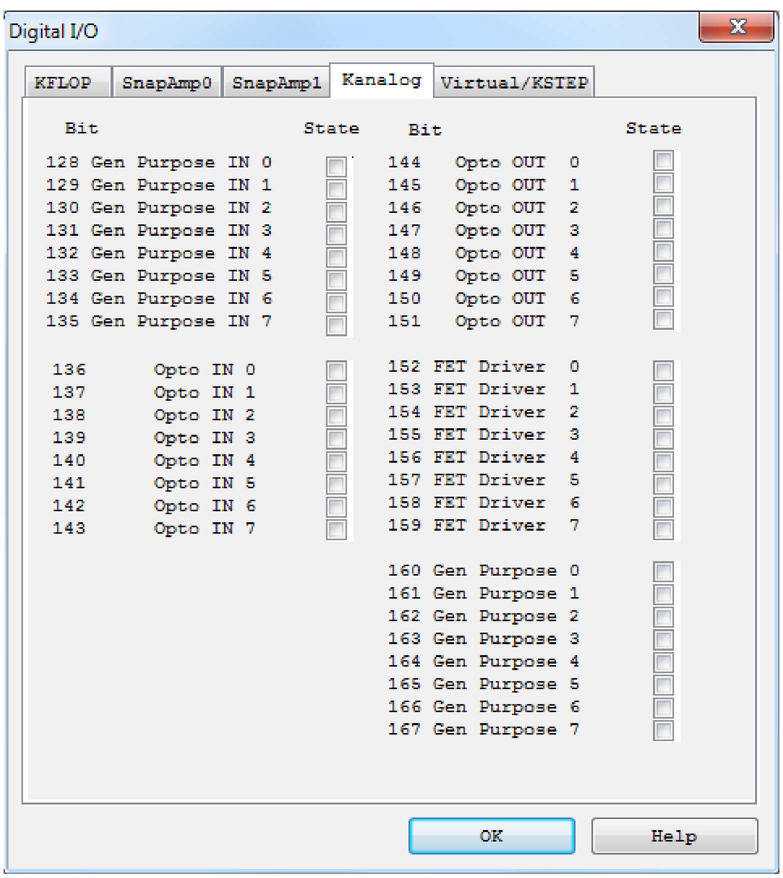
Kogna – Digital I/O Screen

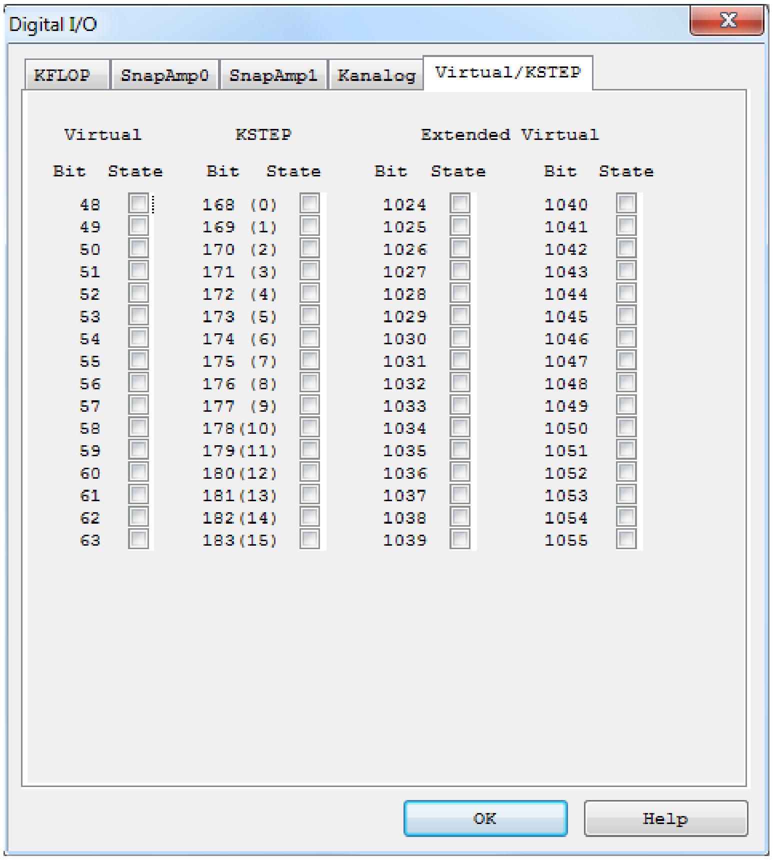


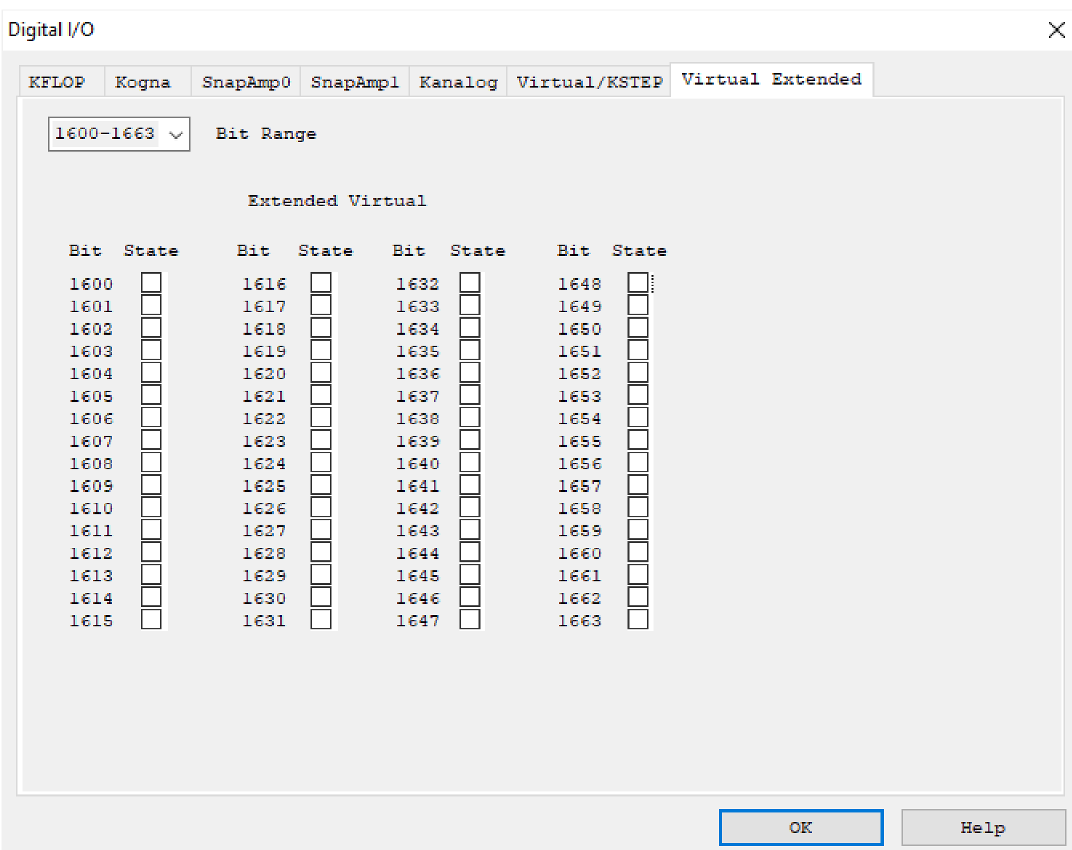

Digital I/O
KFLOP
SnapAmp0
SnapAmp1
Kanalog
Virtual/KSTEP
Output
Bit
State
0 Enc 0 A
1 Enc 0 B
2 Enc 1 A
3 Enc 1 B
4 Enc 2 A
5 Enc 2 B
6 Enc 3 A
7 Enc 3 B
8 Home 0
9 Home 1
10 Home 2
11 Home 3
12 LIM + 0
13 LIM - 0
14 LIM + 1
15 LIM - 1
16 Aux0
17 Aux0
18 Aux0
19 Aux0
20 Aux0
21 Aux0
22 Aux0
23 Aux0
24 Aux0
25 Aux0
26 Aux1
27 Aux1
28 Aux1
29 Aux1
30 Aux1
31 Aux1
32 Aux1
33 Aux1
34 Aux1
35 Aux1
36 Com
37 Com
38 Com
39 Com
40 Com
41 Com
42 Com
43 Com
44
45
46 LED0
47 LED1
OK
Help


Digital I/O
KFLOP
SnapAmp0
SnapAmp1
Kanalog
Virtual/KSTEP
Bit
State
Output
64 Diff 0 Enc 0 A
65 Diff 1 Enc 0 B
66 Diff 2 Enc 1 A
67 Diff 3 Enc 1 B
68 Diff 4 Enc 2 A
69 Diff 5 Enc 2 B
70 Diff 6 Enc 3 A
71 Diff 7 Enc 3 B
72 Opto 0
73 Opto 1
74 Opto 2
75 Opto 3
76 Opto 4
77 Opto 5
78 Opto 6
79 Opto 7
80 GPIO 0
81 GPIO 1
82 GPIO 2
83 GPIO 3
84 GPIO 4
85 GPIO 5
86 GPIO 6
87 GPIO 7
88 GPIO 8
89 GPIO 9
90 GPIO 10
91 GPIO 11
92 GPIO 12
93 GPIO 13
OK
Help


Digital I/O
KFLOP
SnapAmp0
SnapAmp1
Kanalog
Virtual/KSTEP
Bit
State
Output
Bit
State
96 Diff 0 Enc 0 A
97 Diff 1 Enc 0 B
98 Diff 2 Enc 1 A
99 Diff 3 Enc 1 B
100 Diff 4 Enc 2 A
101 Diff 5 Enc 2 B
102 Diff 6 Enc 3 A
103 Diff 7 Enc 3 B
104 Opto 0
105 Opto 1
106 Opto 2
107 Opto 3
108 Opto 4
109 Opto 5
110 Opto 6
111 Opto 7
112 GPIO 0
113 GPIO 1
114 GPIO 2
115 GPIO 3
116 GPIO 4
117 GPIO 5
118 GPIO 6
119 GPIO 7
120 GPIO 8
121 GPIO 9
122 GPIO 10
123 GPIO 11
124 GPIO 12
125 GPIO 13
OK
Help


Digital I/O
KFLOP
SnapAmp0
SnapAmp1
Kanalog
Virtual/KSTEP
Bit
State
Bit
State
128 Gen Purpose IN 0
129 Gen Purpose IN 1
130 Gen Purpose IN 2
131 Gen Purpose IN 3
132 Gen Purpose IN 4
133 Gen Purpose IN 5
134 Gen Purpose IN 6
135 Gen Purpose IN 7
136 Opto IN 0
137 Opto IN 1
138 Opto IN 2
139 Opto IN 3
140 Opto IN 4
141 Opto IN 5
142 Opto IN 6
143 Opto IN 7
144 Opto OUT 0
145 Opto OUT 1
146 Opto OUT 2
147 Opto OUT 3
148 Opto OUT 4
149 Opto OUT 5
150 Opto OUT 6
151 Opto OUT 7
152 FET Driver 0
153 FET Driver 1
154 FET Driver 2
155 FET Driver 3
156 FET Driver 4
157 FET Driver 5
158 FET Driver 6
159 FET Driver 7
160 Gen Purpose 0
161 Gen Purpose 1
162 Gen Purpose 2
163 Gen Purpose 3
164 Gen Purpose 4
165 Gen Purpose 5
166 Gen Purpose 6
167 Gen Purpose 7
OK
Help


Digital I/O
KFLOP
SnapAmp0
SnapAmp1
Kanalog
Virtual/KSTEP
Virtual
KSTEP
Extended Virtual
Bit State
Bit State
Bit State
Bit State
48 168 (0) 1024 1040
49 169 (1) 1025 1041
50 170 (2) 1026 1042
51 171 (3) 1027 1043
52 172 (4) 1028 1044
53 173 (5) 1029 1045
54 174 (6) 1030 1046
55 175 (7) 1031 1047
56 176 (8) 1032 1048
57 177 (9) 1033 1049
58 178(10) 1034 1050
59 179(11) 1035 1051
60 180(12) 1036 1052
61 181(13) 1037 1053
62 182(14) 1038 1054
63 183(15) 1039 1055
OK
Help


Digital I/O
KFLOP
Kogna
SnapAmp0
SnapAmp1
Kanalog
Virtual/KSTEP
Virtual Extended
1600-1663
Bit Range
Extended Virtual
Bit State
Bit State
Bit State
Bit State
1600 1616 1632 1648
1601 1617 1633 1649
1602 1618 1634 1650
1603 1619 1635 1651
1604 1620 1636 1652
1605 1621 1637 1653
1606 1622 1638 1654
1607 1623 1639 1655
1608 1624 1640 1656
1609 1625 1641 1657
1610 1626 1642 1658
1611 1627 1643 1659
1612 1628 1644 1660
1613 1629 1645 1661
1614 1630 1646 1662
1615 1631 1647 1663
OK
Help

The ***Digital I/O Screen*** displays and allows changes to the current state of ***Kogna*** digital I/O bits. Digital IO bits are present on the ***Kogna*** as well as any additional option cards (such as with two SnapAmps or Kanalog as shown above) as well as Virtual Bits (simulated with memory). IO bits are numbered 0-2047.

Kogna 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 many bits that may be utilized as GPIO (bits 0 - 45). Some bits may be independently defined as either an input or an output. On Power UP ***Kogna*** defines all I/O as inputs by default. A bit may be configured as an output by checking the corresponding box in the "Output" columns where available. 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 ***Kogna***.

Kogna bits 0-15 are connected to the JP7 connector normally which may be used by Kogna 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.

Kogna 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 Kogna (See ReadBit(), SetBit(), ClearBit(), or SetStateBit()) or by Script commands (See ReadBit, SetBit, ClearBit, or SetStateBit) sent to Kogna.

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 ***Kogna*** Hardware Connector Descriptions, SnapAmp Hardware Connector Descriptions, Kanalog Hardware Connector Descriptions, or KStep Hardware Connector Descriptions for which IO bits are connected to the various connectors on those optional boards.

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.

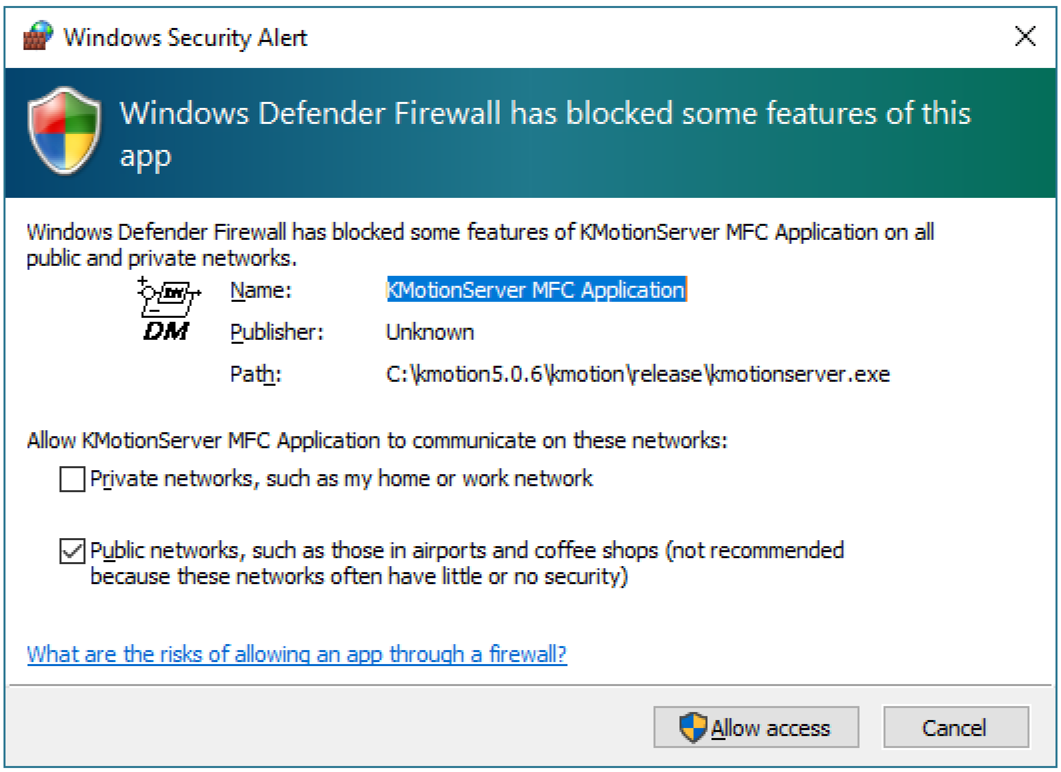
Digital I/O bits 46 and 47 are dedicated to the control of the two LED's on ***Kogna.*** These two outputs are configured as outputs and tuned on when ***Kogna*** powers up. The are available for use as User status if desired.

For other IO bits and any dedicated purposes or modes see the Kogna Connector documentation.

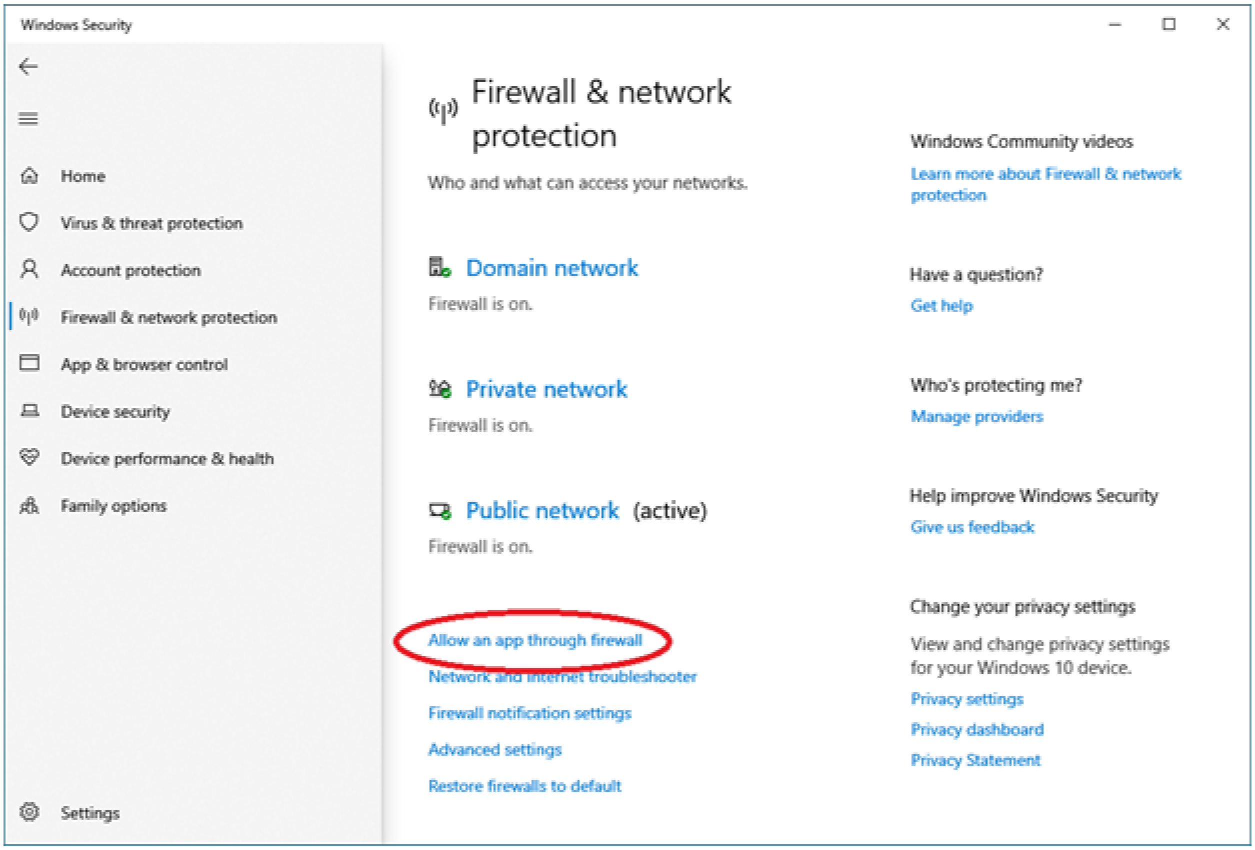
Kogna IP Address Discovery

KMotionServer periodically does a UDP multicast to 239.81.92.240 port 25000. All Kognas join this multicast group and so should recieve any packet sent to this multicast address. If received by Kogna (and is a proper 64 byte packet beginning with "Kogna?") the Kogna will reply to the source (port 25001 also 64 byte packet) with "I am Kogna SNXXX" where XXX is the Kogna's Serial Number. Our Apps default to connect to any board discovered (ID=0) but can also be told to connect to a specific Kogna Serial Number or specific IP Address. Once the IP address of the Kogna is discovered by the Host, the Host opens TCP/IP Port 2000 to send commands and data.

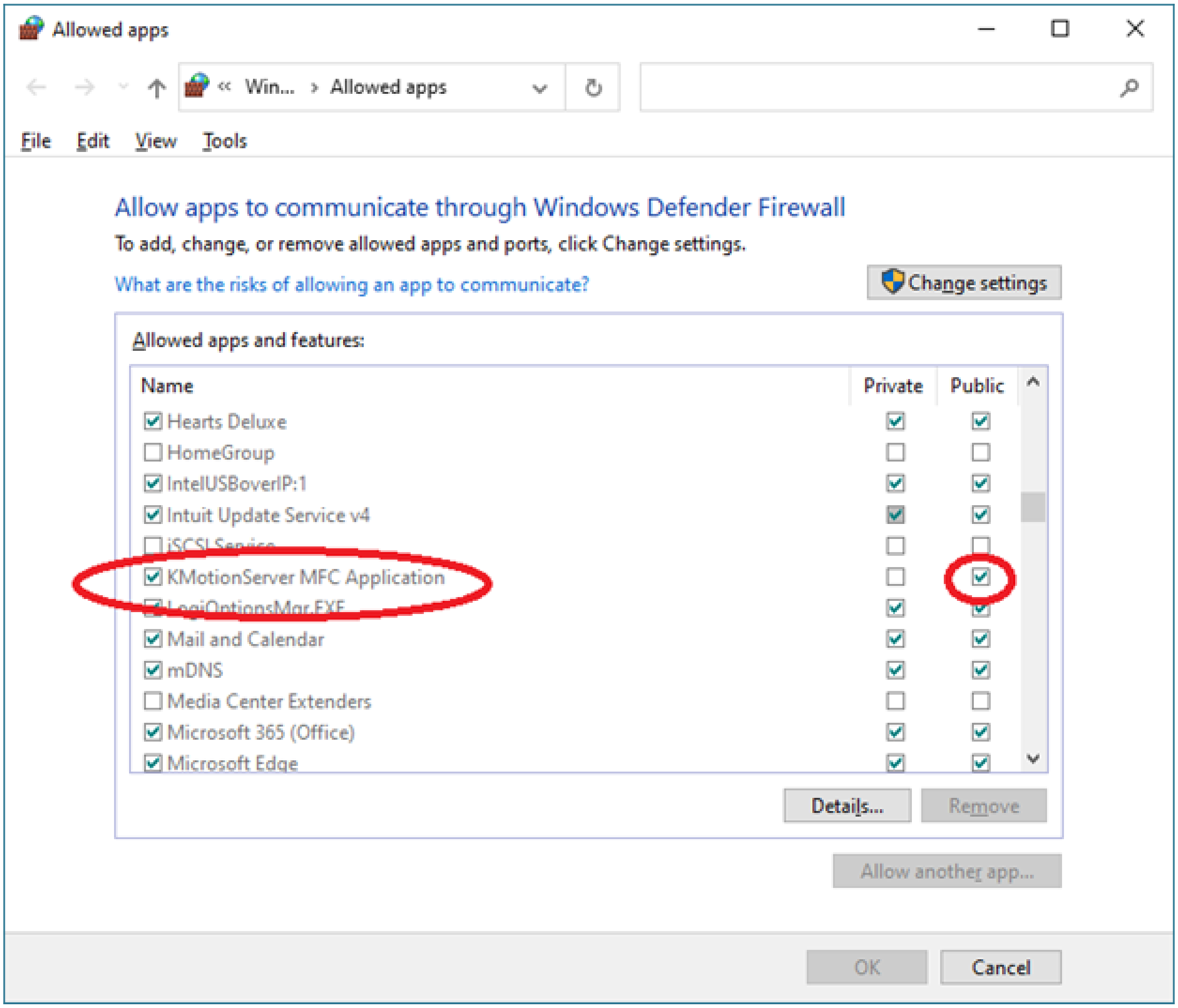
In order to allow KMotion Apps to autodiscover Kognas it may be necessary to turn off your Firewall. Or as shown below an exception may be required for KMotionServer.exe.



If the exception wasn't allowed through the above dialog, it can be added or modified in Windows Firewall settings.



Check that KMotionServer is allowed and has Public network access. If more than one installation has been made, there may be more than one. Check "Details" to see where it is located. If not present, use "Allow another app…" to add it. **Note:** to make changes, first push the "Change Settings" button.


Allowed apps
Win... > Allowed apps
File Edit View Tools
Allow apps to communicate through Windows Defender Firewall
To add, change, or remove allowed apps and ports, click Change settings.
What are the risks of allowing an app to communicate?
Change settings
Allowed apps and features:
Name
Private
Public
Hearts Deluxe
HomeGroup
IntelUSBoverIP:1
Intuit Update Service v4
iSCSI Service
KMotionServer MFC Application
LogiOptionsMgr.EXE
Mail and Calendar
mDNS
Media Center Extenders
Microsoft 365 (Office)
Microsoft Edge
Details...
Remove
Allow another app...
OK
Cancel

KFLOP Summary

8-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.

Quick Start

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.

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.

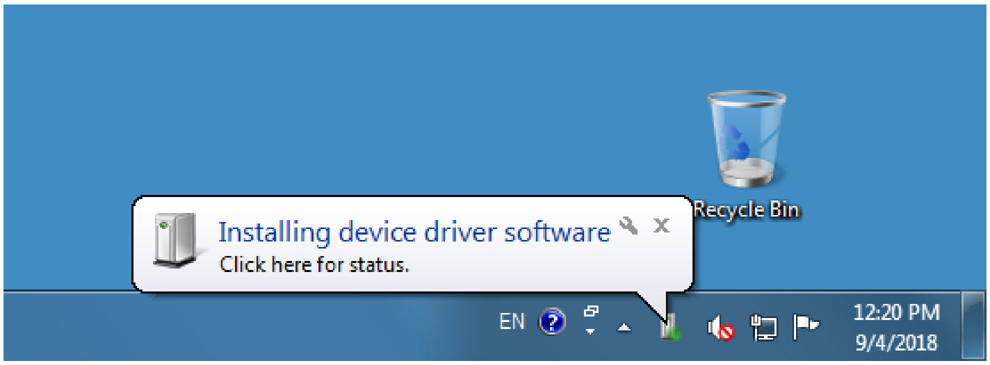
Note: it is NOT recommended to use the same power supply that is powering your PC and Hard drives to power the KFLOP. Motor noise and power surges have the possibility to cause damage or loss of data within the PC.

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.

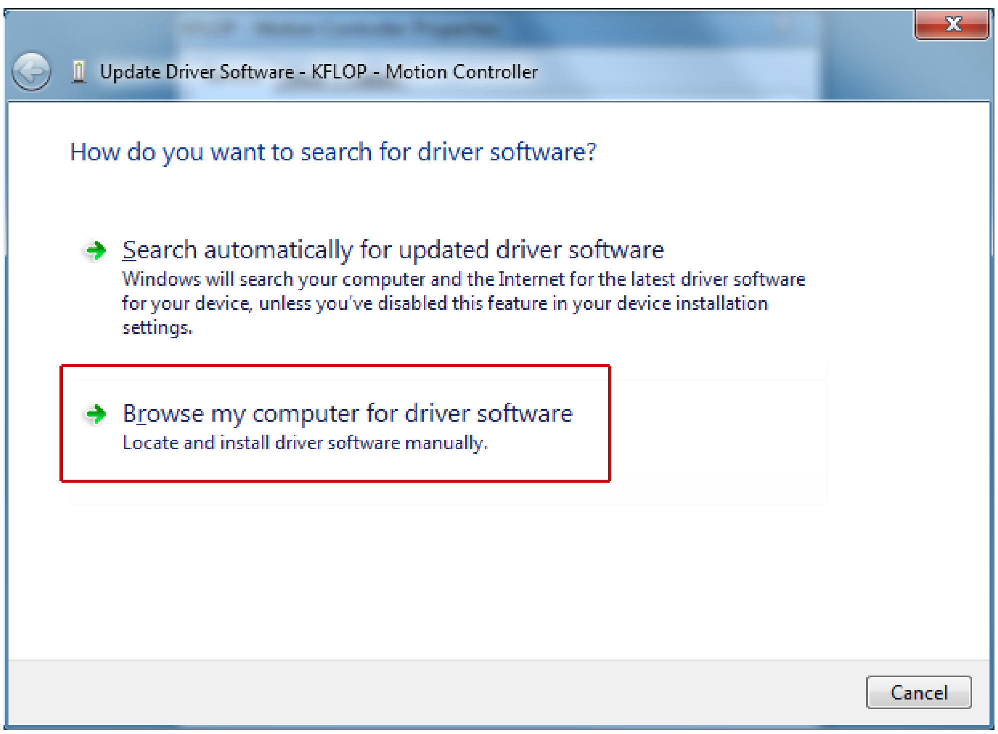
Now connect a USB cable from ***KFLOP*** to your PC and follow the USB Driver instructions below.

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.

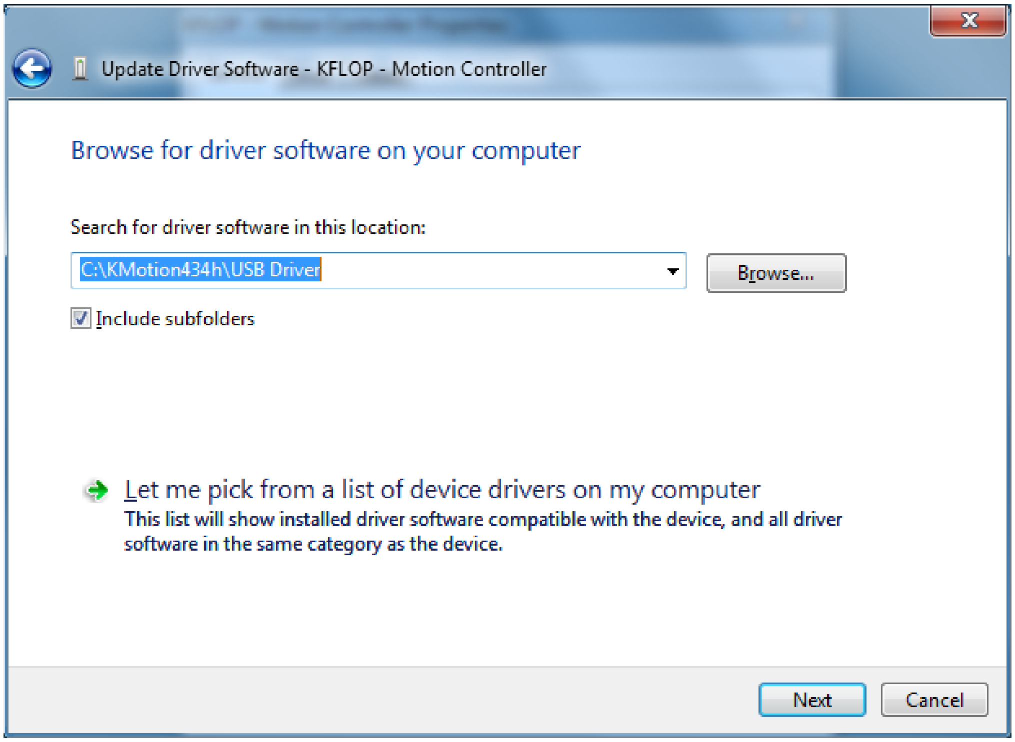


Select "**Browse my computer for driver software".**

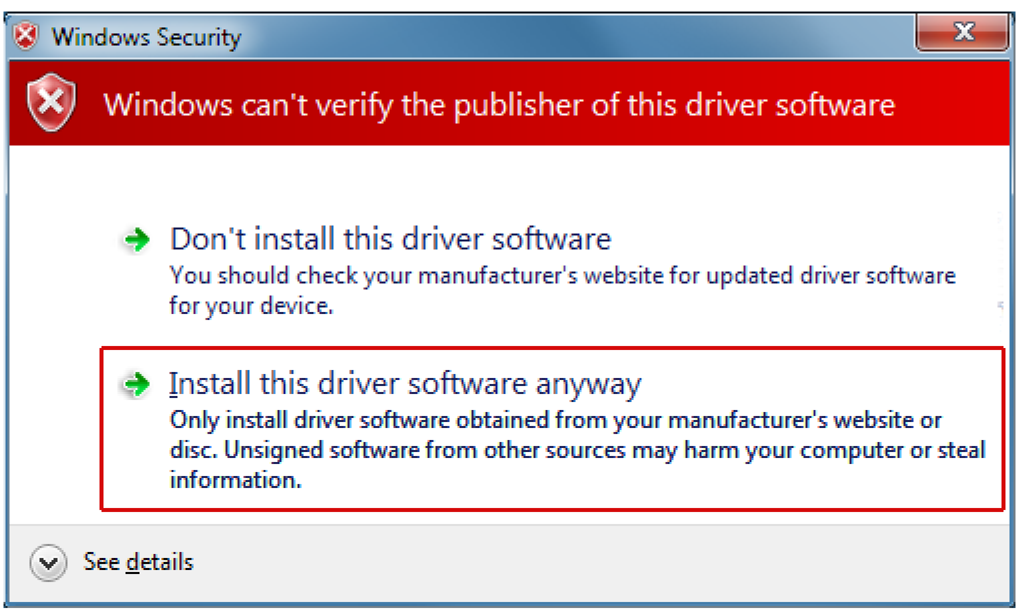


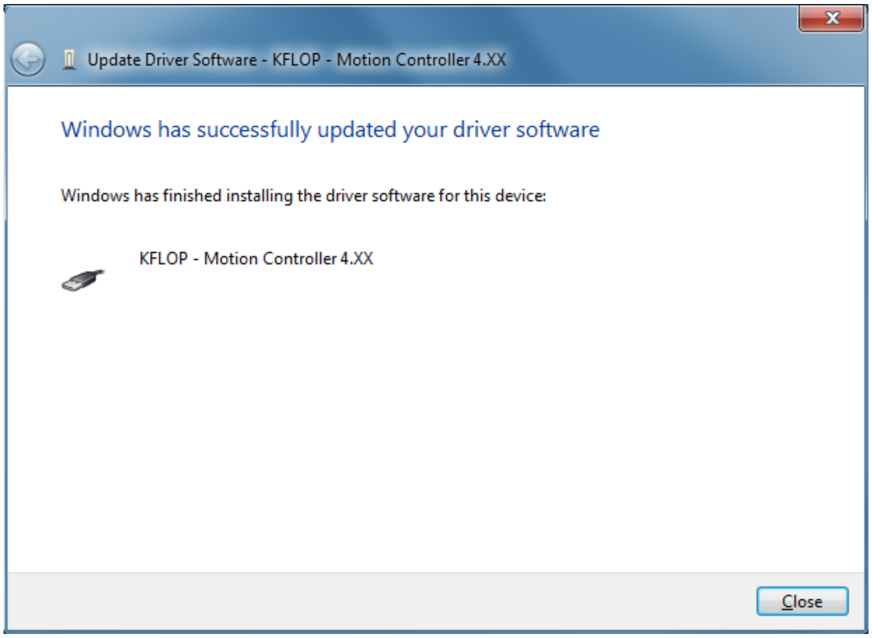
If you haven't already, download and install the complete ***KFLOP*** Software including drivers available at: http://dynomotion.com/Software/Download.html

Browse to within the subdirectory where you selected the ***KFLOP*** Software to be installed to the "**USB DRIVER**" subdirectory. (If the sotware was installed into the default subdirectory, the location would be: **C:\KMotion4xx\USB DRIVER)**.



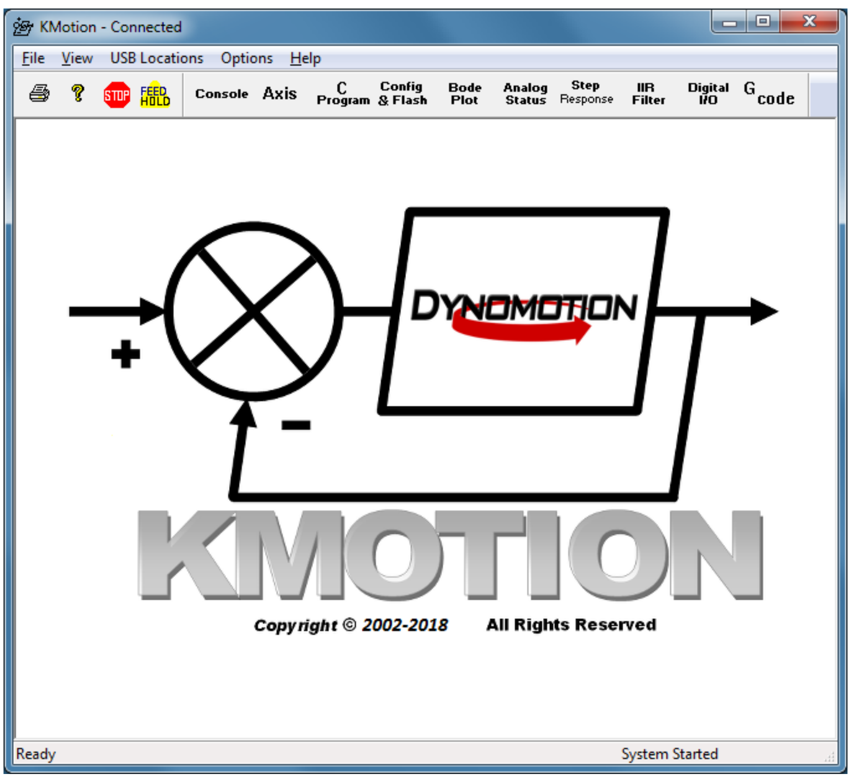
Note: If your system has not received the latest Windows™ security updates, you might get a warning screen. Select "**Install this driver software anyway"**.



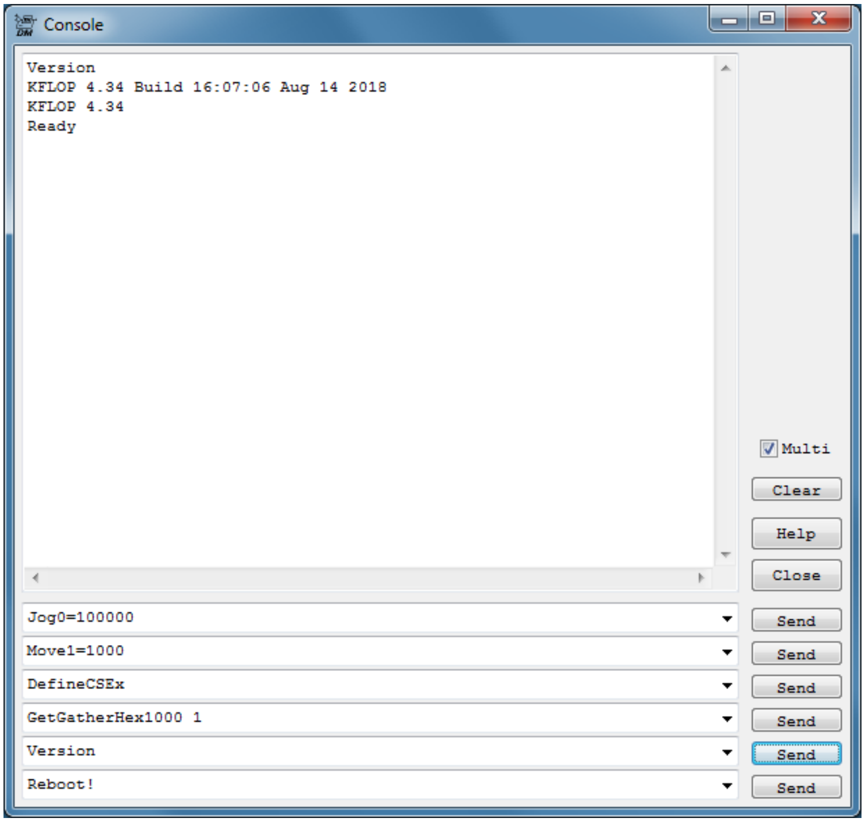


If successful, the above screen should appear. Click **"Close".**

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.

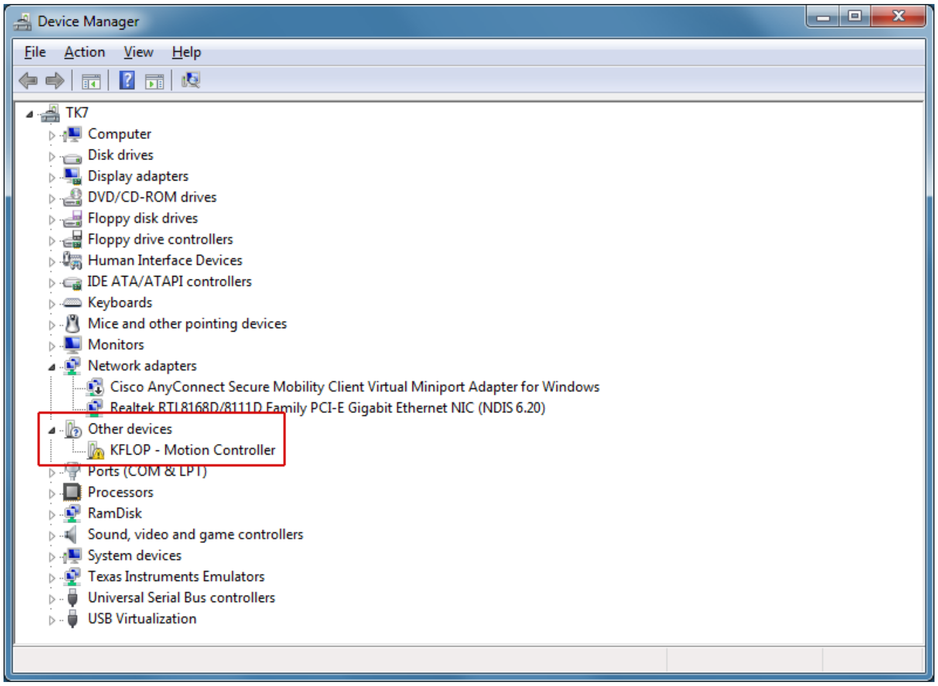


At the main tool bar select the "Console" button to Display the Console Screen. Enter the "Version" command and press "Send". An output message will appear indicating successful communication between the PC and KFLOP.

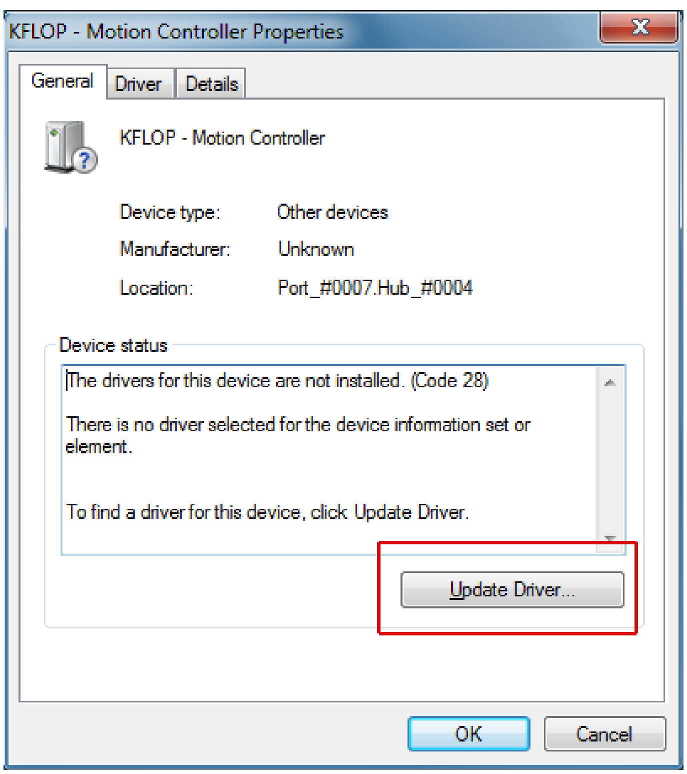


If you are updating to a new version of KMotion and the driver versions do not match or you have any other problem related to the drivers, follow the steps below:

Go to the **Device Manager** and right-mouse-click on "KFLOP Motion Controller" under "Other devices" and select **"Update Driver Software"**.



On the next screen click **"Update driver"** and follow the steps described above.



If the wrong or no drivers were installed, find KFLOP in the Device Manager and right-mouse-click**"Uninstall"**. On the next screen, select the checkbox next to **"Delete the driver software for this device"**.

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!!!

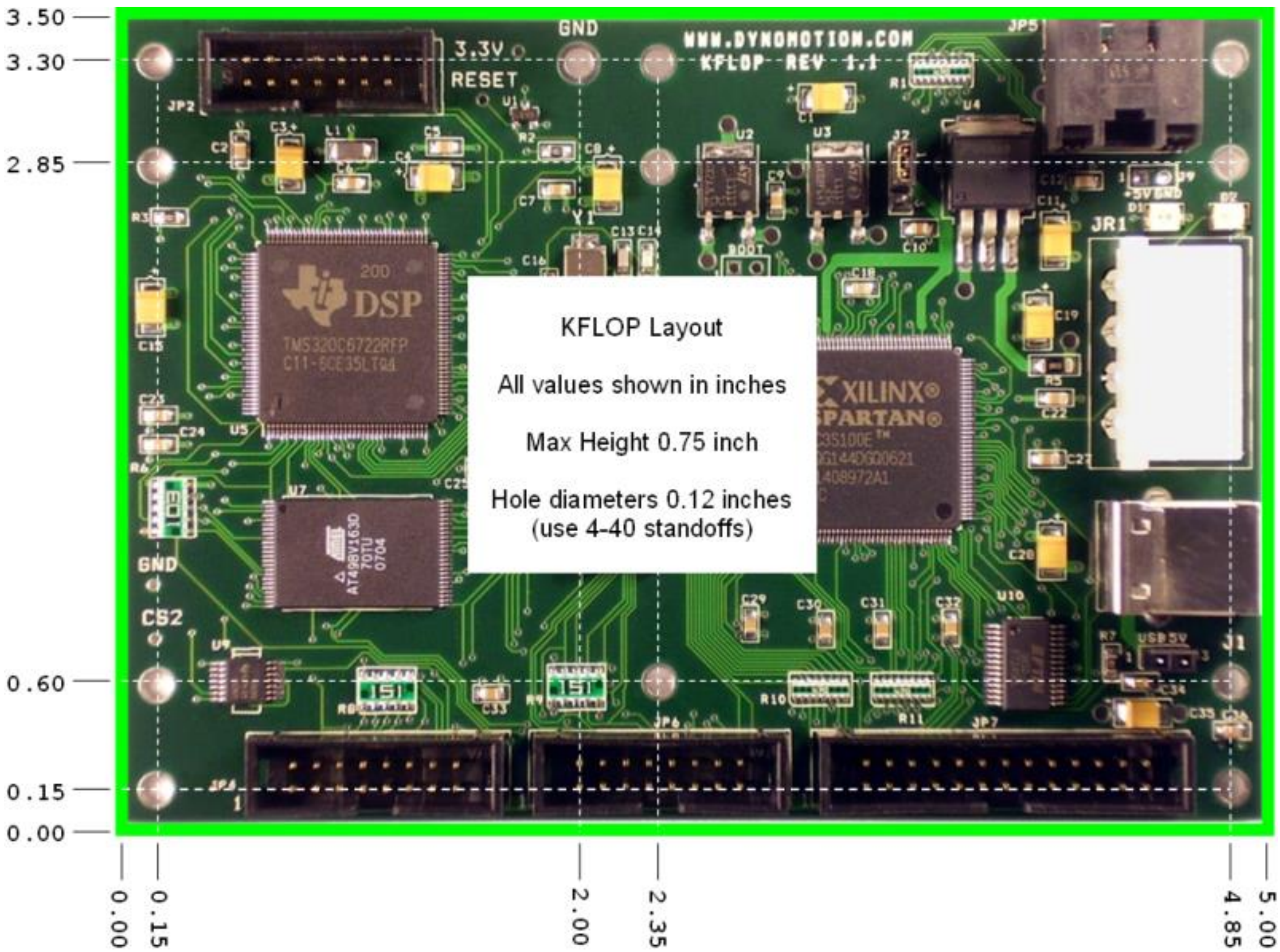
KFLOP Hardware Specification

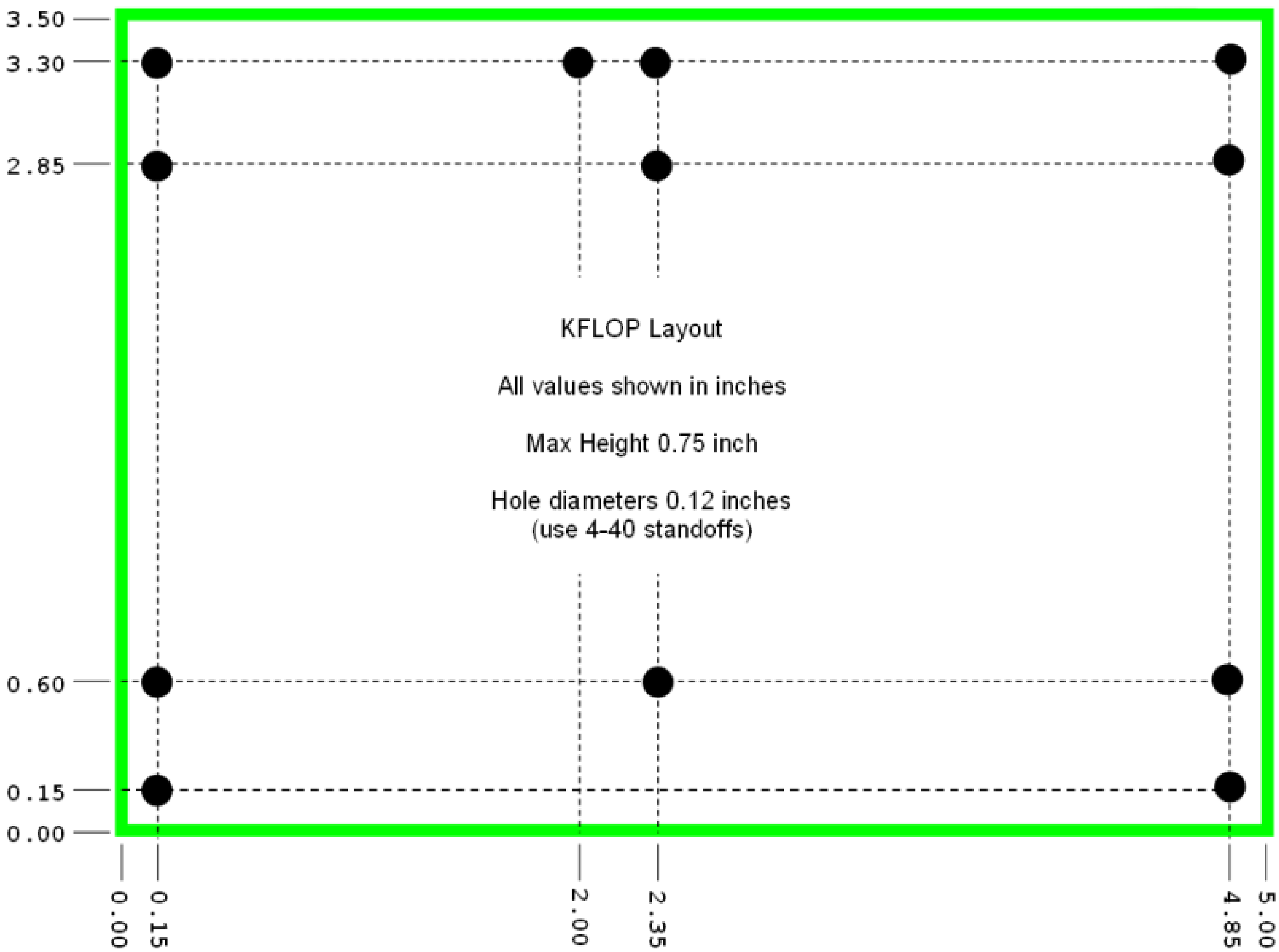
Function	Parameter	Specification
Processor	CPU Memory	TMS320C67-200MHz DSP 1.2GFLOP 32/64-Bit Native Floating Point FLASH 2 MBytes SDRAM 16 Mbytes
Interface	Host	USB 2.0 Full Speed
Connectors	I/O General Purpose I/O Com Aux#0 IO Aux#1 IO USB System Power	26 pin Header 8 pin RJ45 16 pin Header 16 pin Header Type B Molex 4-pin (Disk drive type)
Servo Loop	Sample Rate Compensation Feed Forward	90µs 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 ±10% 2.5A 0.35 A
User I/O	Digital Encoders	45 Gen Purpose LVTTL (24 are 5V Tolerant) (4) single-ended, 1 MHz
Environment	Operating Temperature Storage Temperature Humidity	0-40º C 0-40º 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 Specification

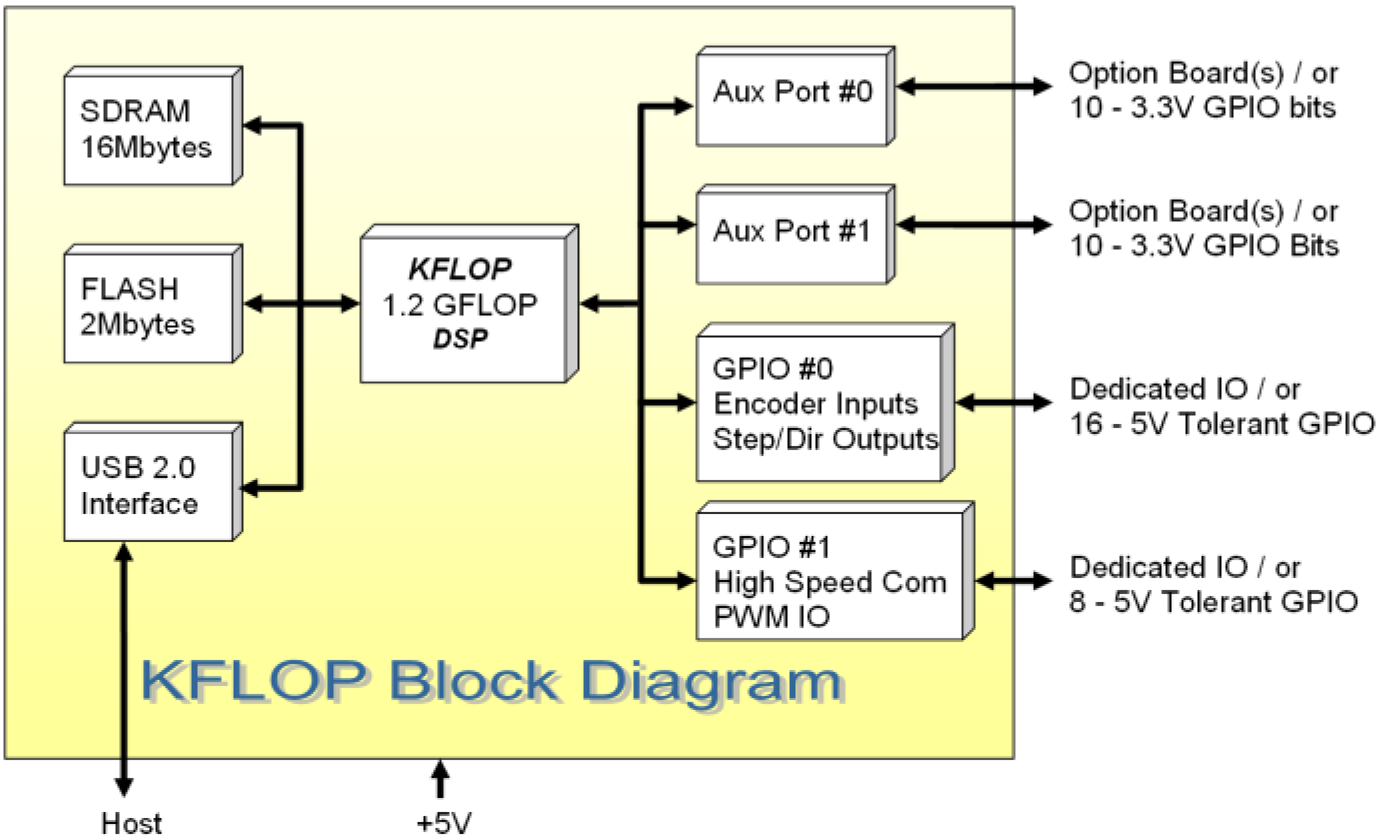
Function	Parameter	Specification
User Programs	Language Number concurrent Stand alone mode	C 7 Yes
Host Requirements	OS Interface	MS Windows™ 7 through 11 USB 2.0
Interface Library	Multi-Thread Multi-Process Multi-Board MS Windows™ VC++ MS Windows™ VB	Yes Yes Yes Supported Supported
C Compiler	TCC67	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

KFLOP Board Layout

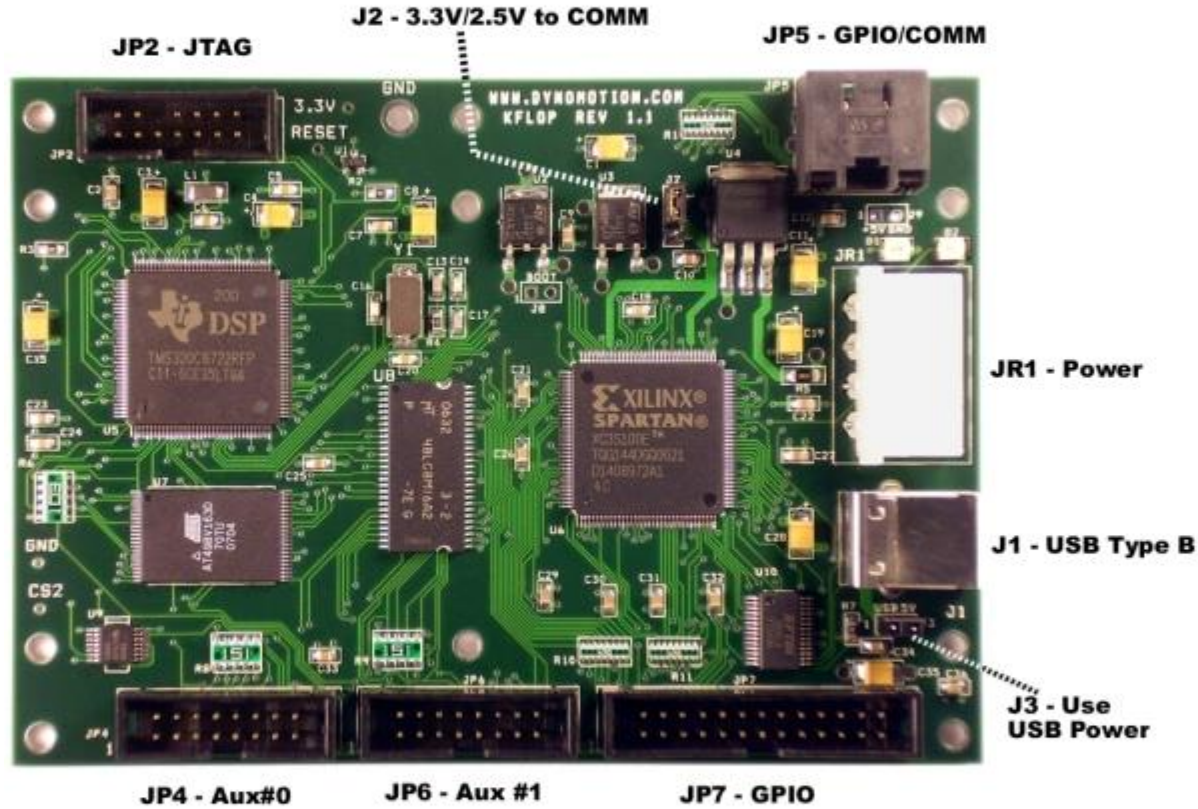



3.50
3.30
2.85
0.60
0.15
0.00
0.00
0.15
2.00
2.35
4.85
5.00
KFLOP Layout
All values shown in inches
Max Height 0.75 inch
Hole diameters 0.12 inches
(use 4-40 standoffs)

Block Diagram



KFLOP - Connector Pinouts

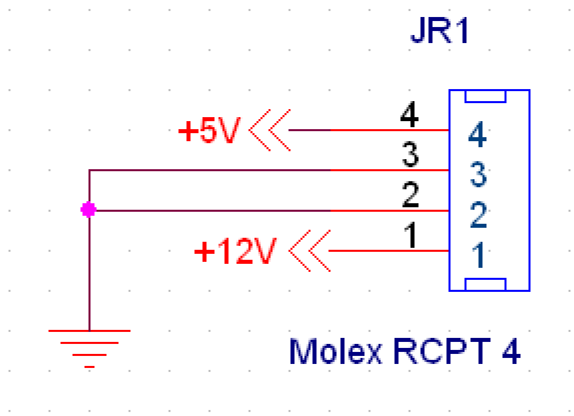


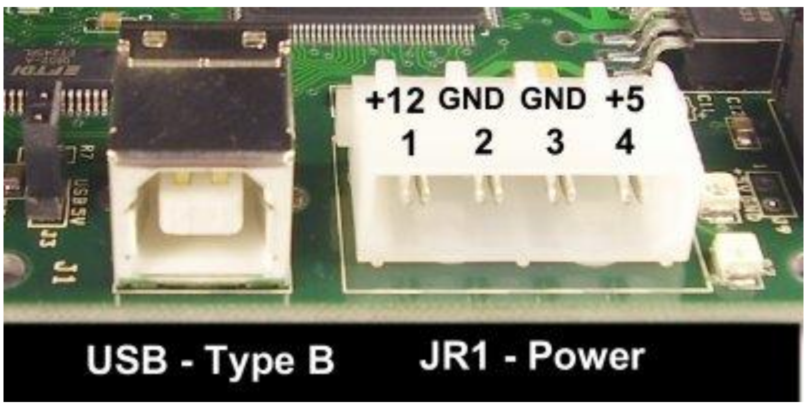
JR1 - +5V Power (regulated +/- 5%)

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.

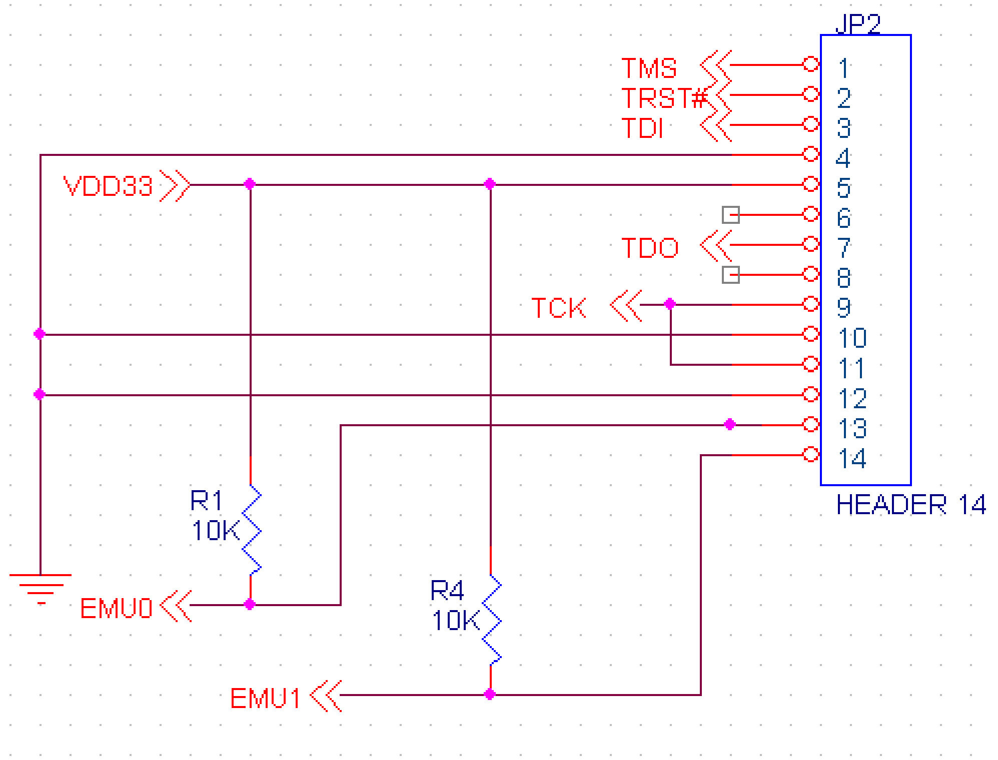
Under some conditions USB power may be used to avoid requiring an external +5V power supply connection. USB specifies a maximum of 500ma may be drawn from the USB cable. KFLOP itself consumes 350ma. If external IO draws less than the remaining 150ma USB supplied power may be used. To utilize USB power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





JP2 - JTAG

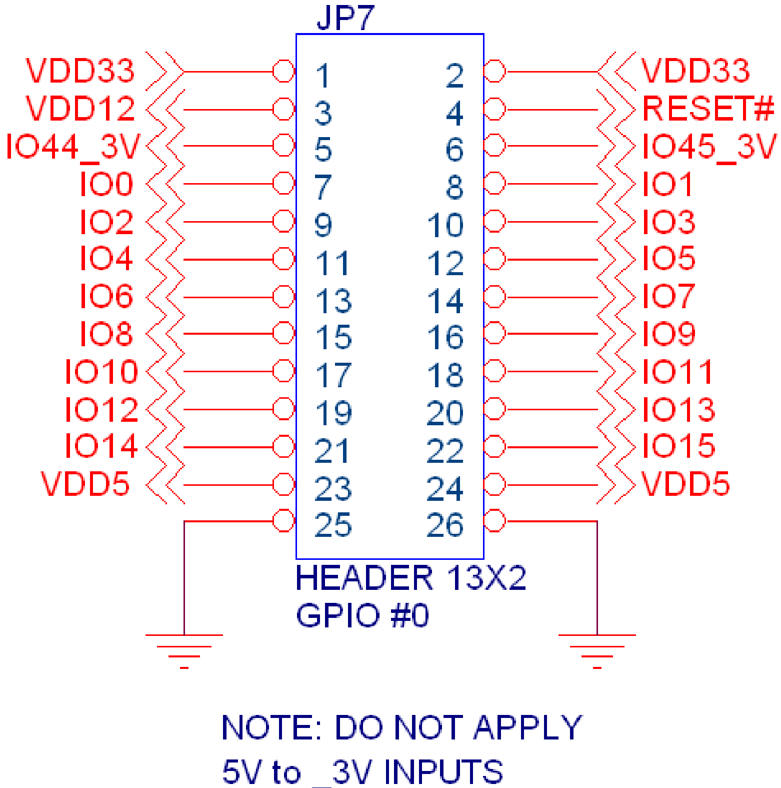
This connector is only used for advanced debugging using an XDS510 JTAG type in circuit emulator. A small amount of regulated 3.3V (<0.5 Amp) is available on this connector if needed for external use.

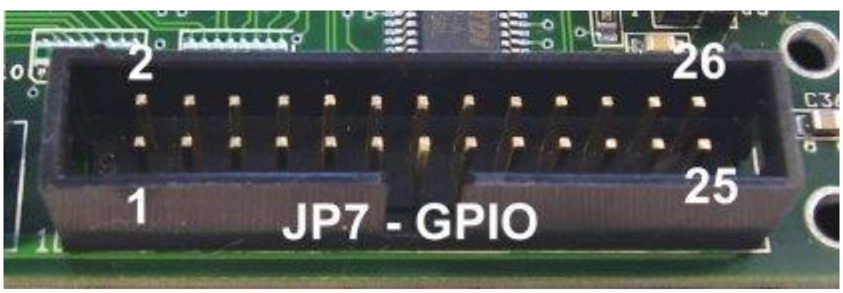


JP7 - Digital IO

18 LVTTL 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 LVTTL (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!


JP7
VDD33
VDD12
IO44_3V
IO0
IO2
IO4
IO6
IO8
IO10
IO12
IO14
VDD5
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
VDD33
RESET#
IO45_3V
IO1
IO3
IO5
IO7
IO9
IO11
IO13
IO15
VDD5
HEADER 13X2
GPIO #0
NOTE: DO NOT APPLY
5V to _3V INPUTS

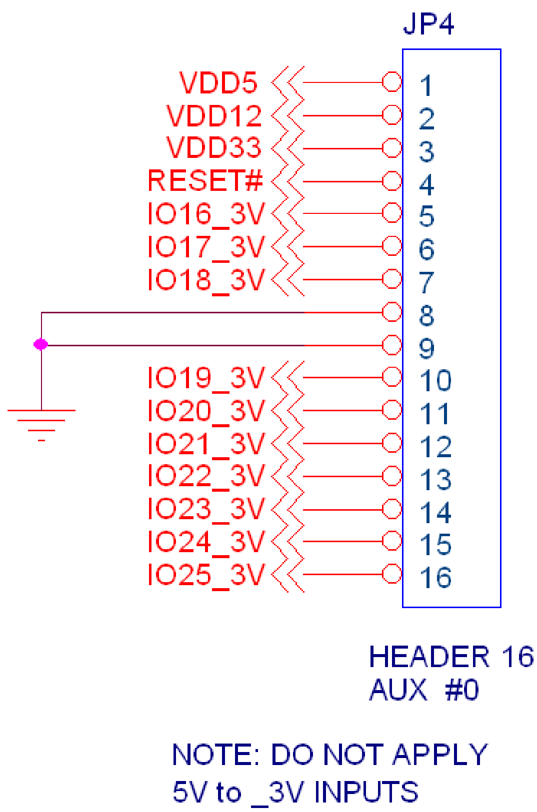

2
26
1
25
JP7 - GPIO

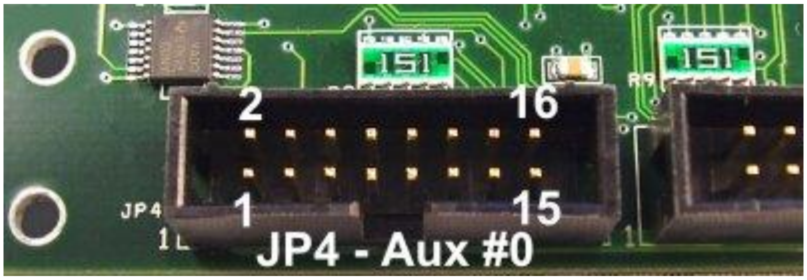
Pin	Name	Description
1	VDD33	+3.3 Volts Output
2	VDD33	+3.3 Volts Output
3	VDD12	+12 Volts Output
4	RESET#	Power up Reset (low true) output
5	IO44	Gen Purpose LVTTL I/O (3.3V Only)
6	IO45	Gen Purpose LVTTL I/O (3.3V Only)
7	IO0	Gen Purpose LVTTL I/O (5V Tolerant) or Axis 0 Encoder Input Phase A
8	IO1	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

Note: Homes and Limits are recommendations only. Any input may be used.

JP4 - Aux Connector #0

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. The first 8 IO (IO16-IO23) contain 150ohm termination resistors (pull downs).

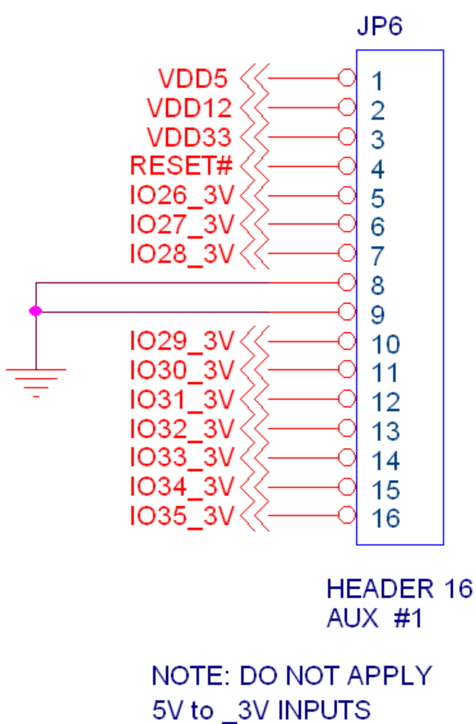


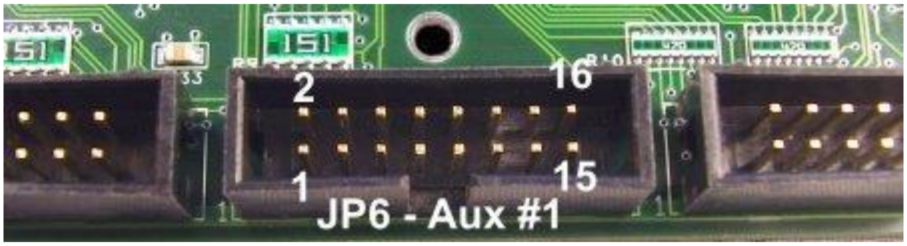


Pin	Name	Description
1	VDD5	+5 Volts Output
2	VDD12	+12 Volts Output
3	VDD33	+3.3 Volts Output
4	RESET#	Power up Reset (low true) output
5	IO16	Gen Purpose LVTTL I/O (3.3V Only)
6	IO17	Gen Purpose LVTTL I/O (3.3V Only)
7	IO18	Gen Purpose LVTTL I/O (3.3V Only)
8	GND	Digital Ground
9	GND	Digital Ground
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

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. The first 8 IO (IO26-IO33) contain 150ohm termination resistors (pull downs).

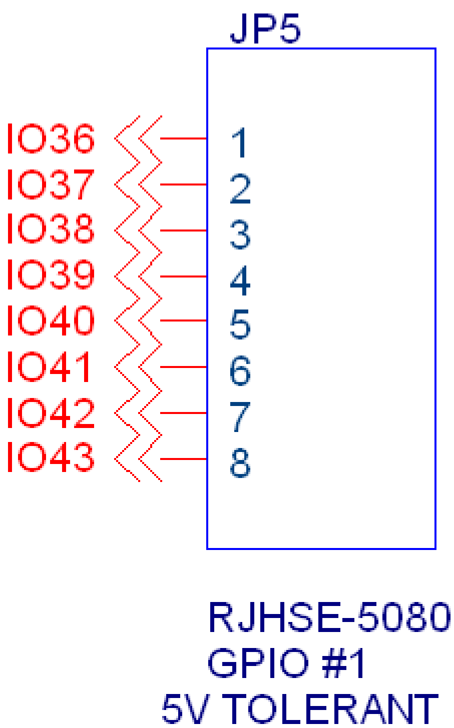


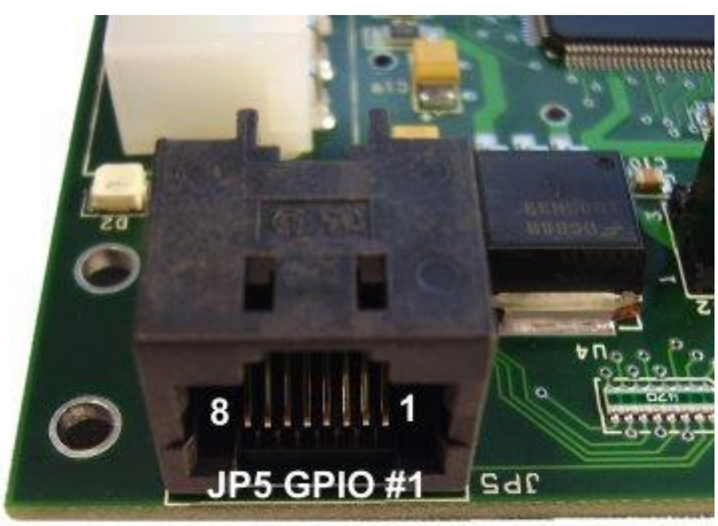


Pin	Name	Description
1	VDD5	+5 Volts Output
2	VDD12	+12 Volts Output
3	VDD33	+3.3 Volts Output
4	RESET#	Power up Reset (low true) output
5	IO26	Gen Purpose LVTTL I/O (3.3V Only) or PWM0 out
6	IO27	Gen Purpose LVTTL I/O (3.3V Only) or PWM1 out
7	IO28	Gen Purpose LVTTL I/O (3.3V Only) or PWM2 out
8	GND	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

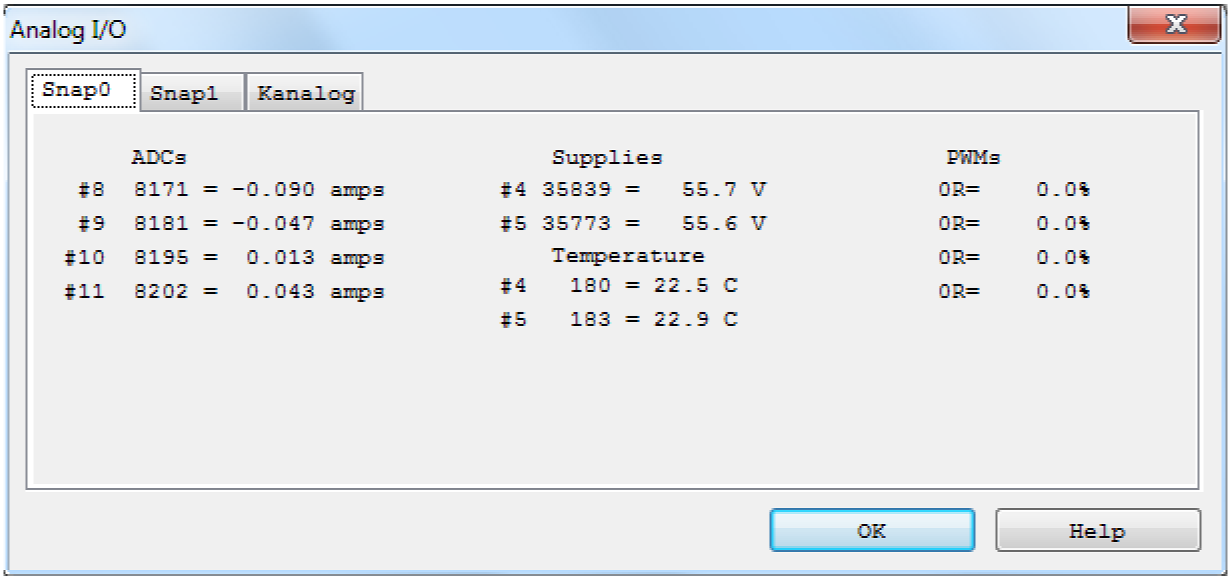
Low Voltage Differential RJ45 Connector. This connector is intended for high-speed low-voltage differential communication over a twisted pair cable use. However it may also be used as 8 General Purpose digital I/O (LVTTL 5V Tolerant)

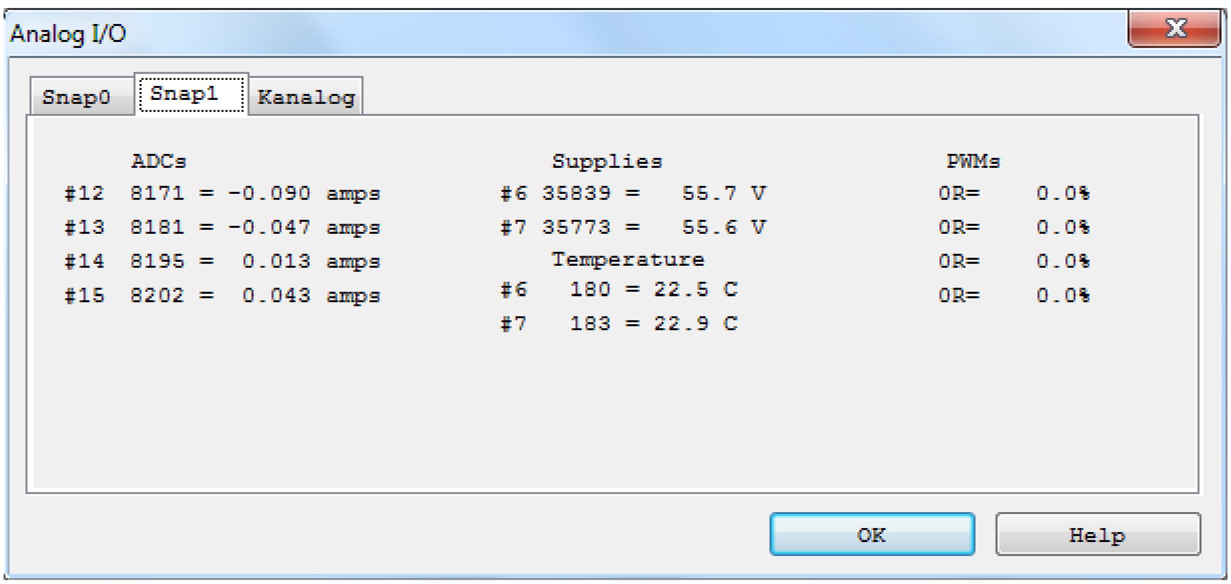


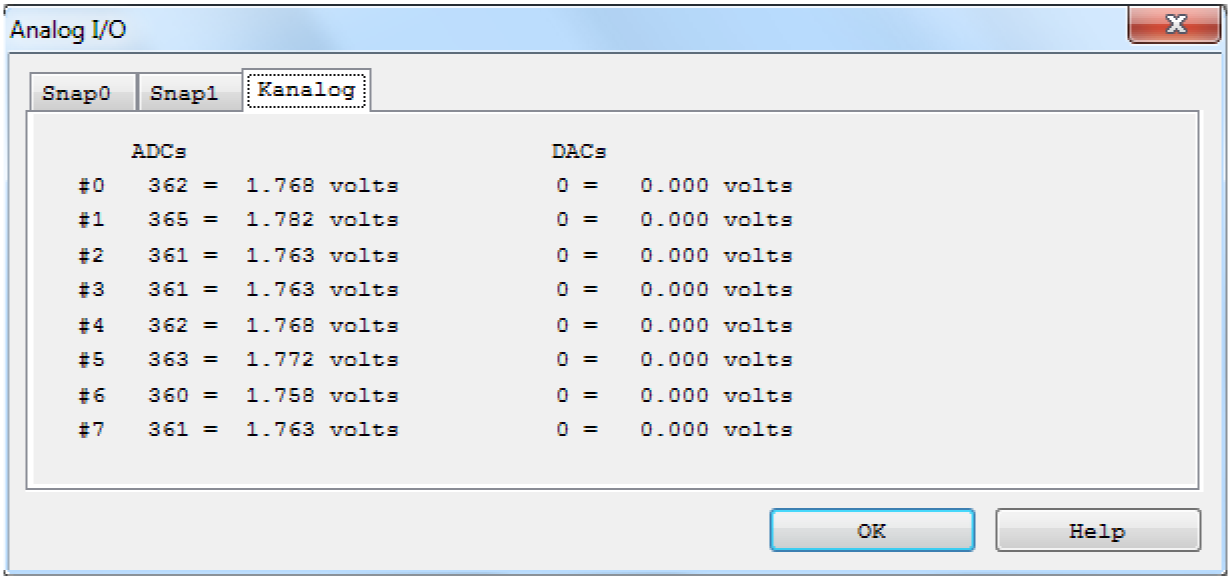


Pin	Name	Description
1	IO36	Gen Purpose LVTTL I/O (5V Tolerant) or (or Step 4 output) or (Axis 4 Encoder Input Phase A)
2	IO37	Gen Purpose LVTTL I/O (5V Tolerant) or (or Dir 4 output) or (Axis 4 Encoder Input Phase B)
3	IO38	Gen Purpose LVTTL I/O (5V Tolerant) or (or Step 5 output) or (Axis 5 Encoder 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







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.

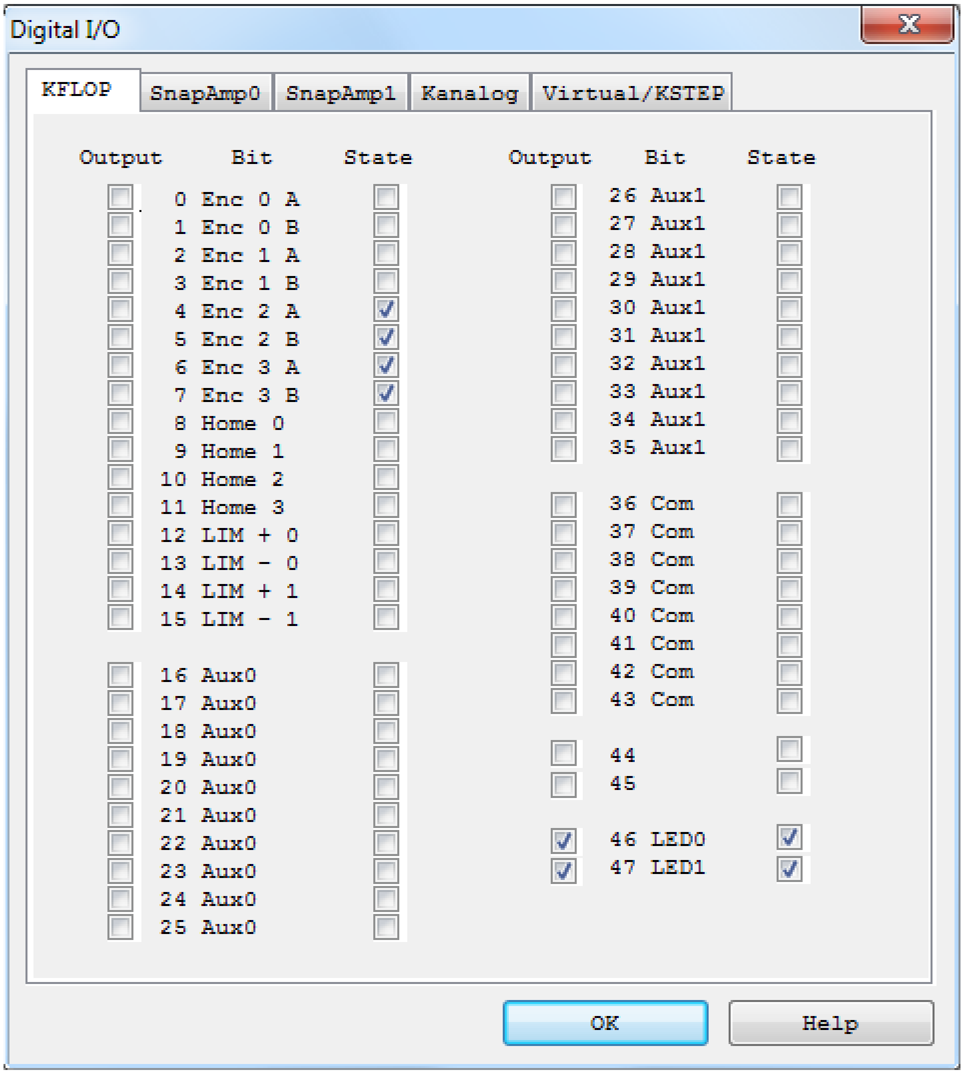
A PWM channel may also operate in current loop mode. In current loop mode the PWM duty cycle is automatically adjusted in order to maintain the commanded current. The PWM channel status will show the command value in counts (-4095 ... +4095), followed by a "C", and the percent of full scale.

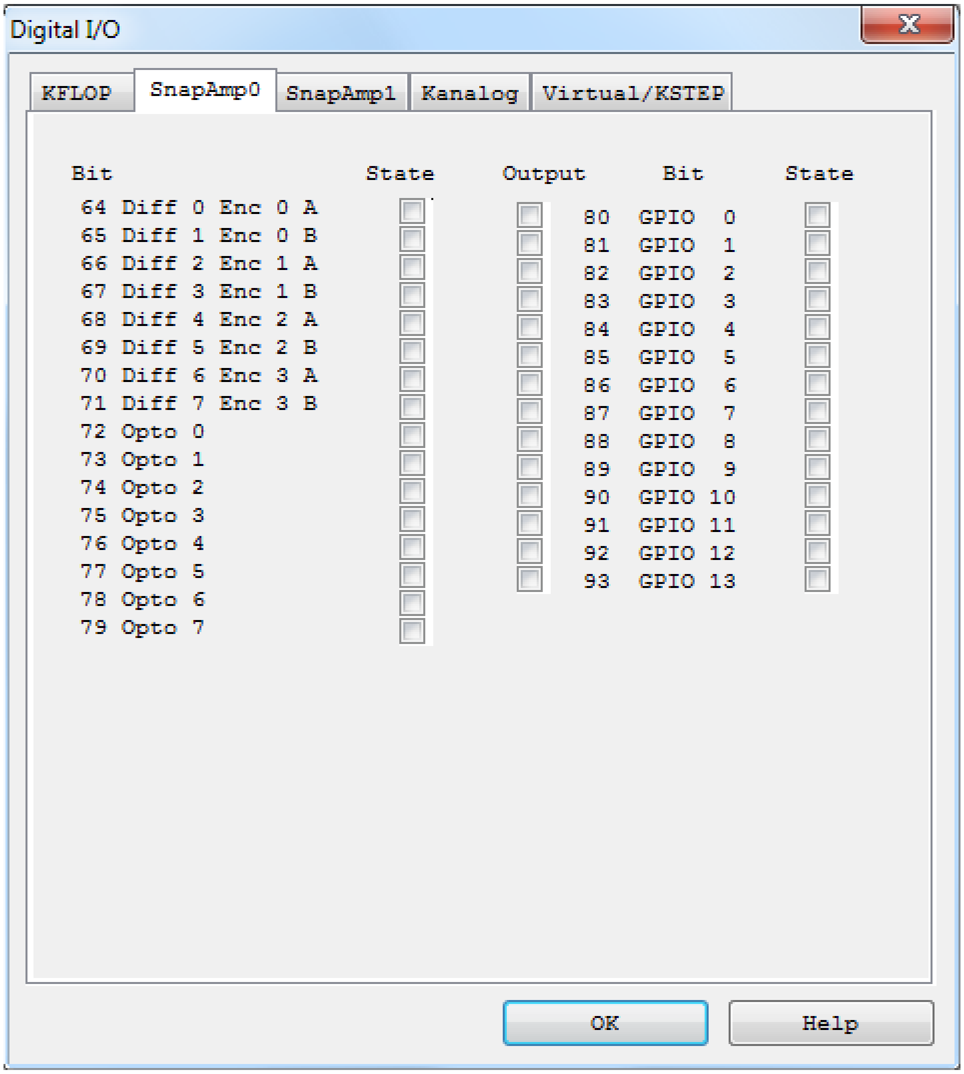
If a pair of PWM channels is operating in 3 Phase mode, the PWM channel status will show the value in counts (-230 ... 230) after the first PWM channel and the phase angle in degrees after second PWM channel.

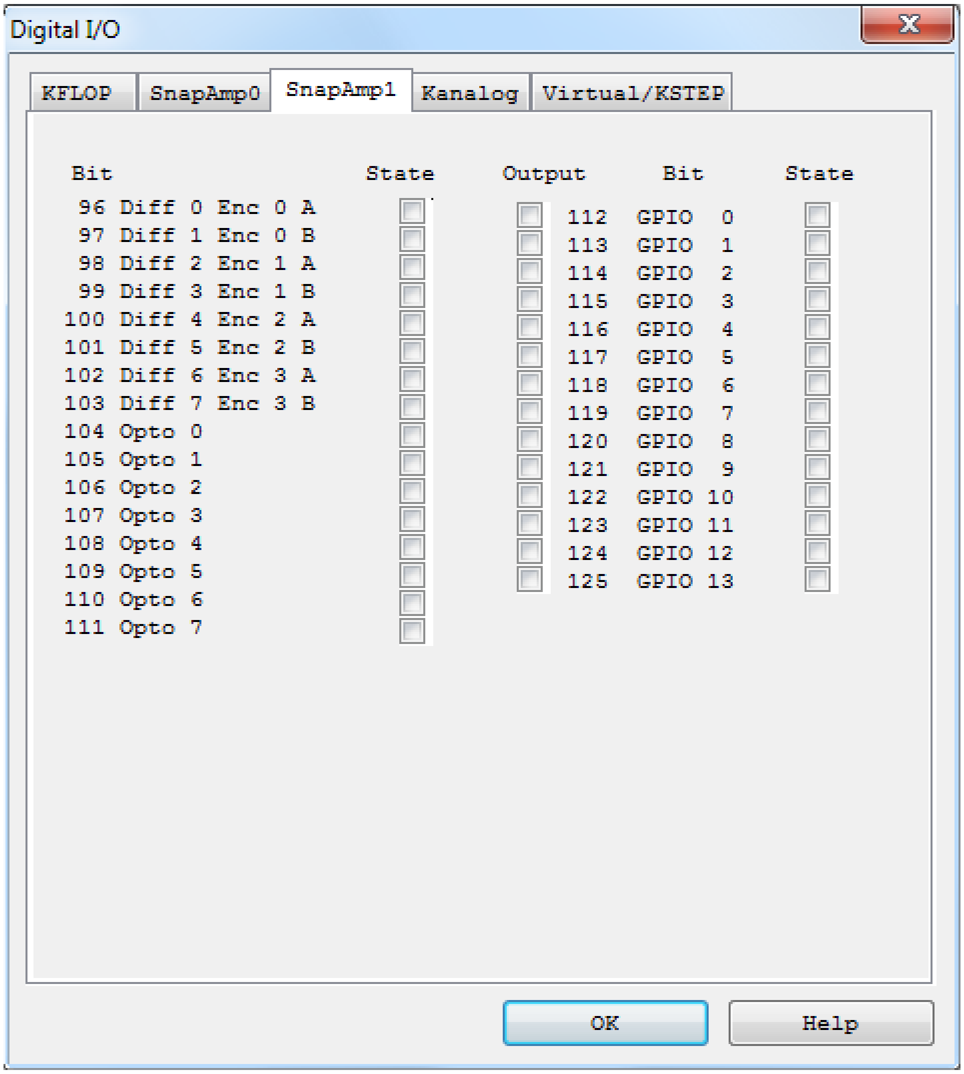
The example status below shows PWM channels 8 and 9 operating in 3 phase mode, PWM channel 10 operating in Normal mode, and PWM channels 11 operating in Recirculating mode.

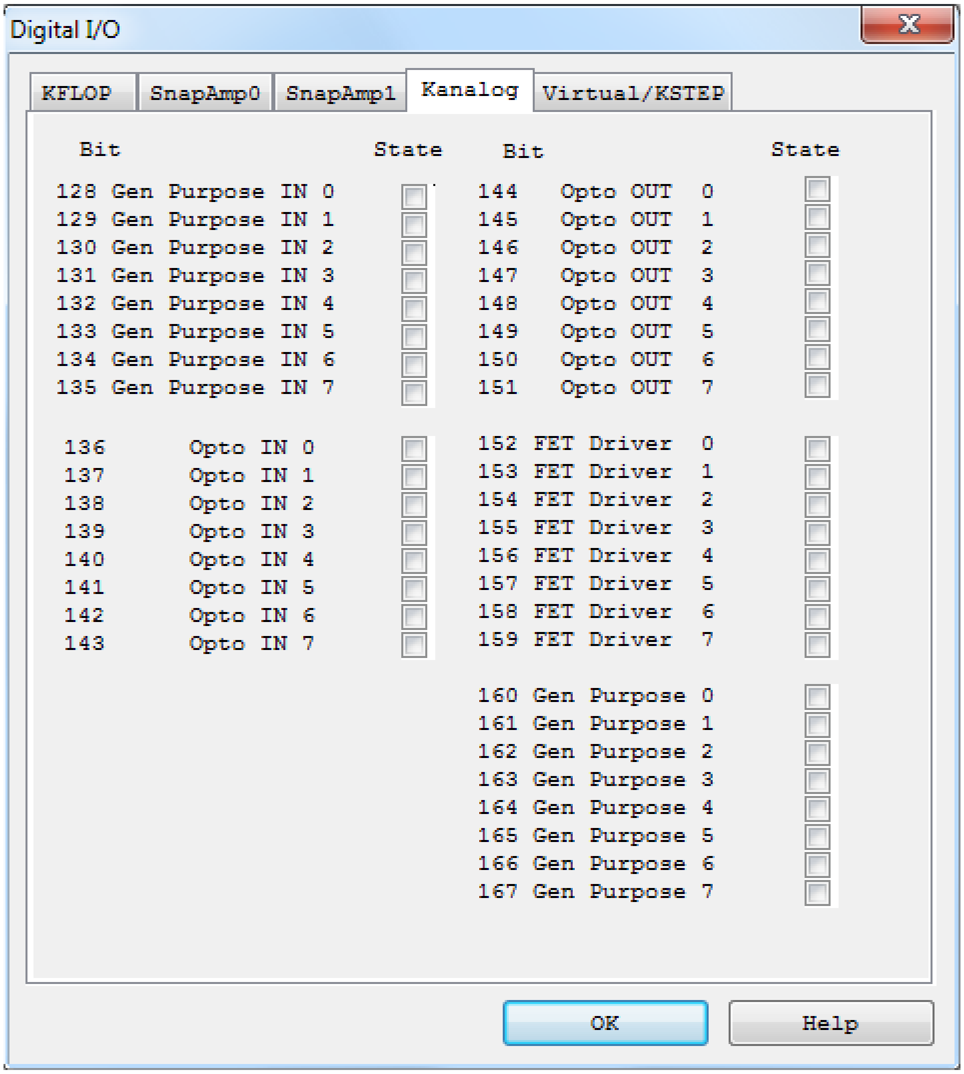
```
   PWMs
100 =   43.5%
  120 degrees
 50 =   19.5%
200R=   39.1%
```

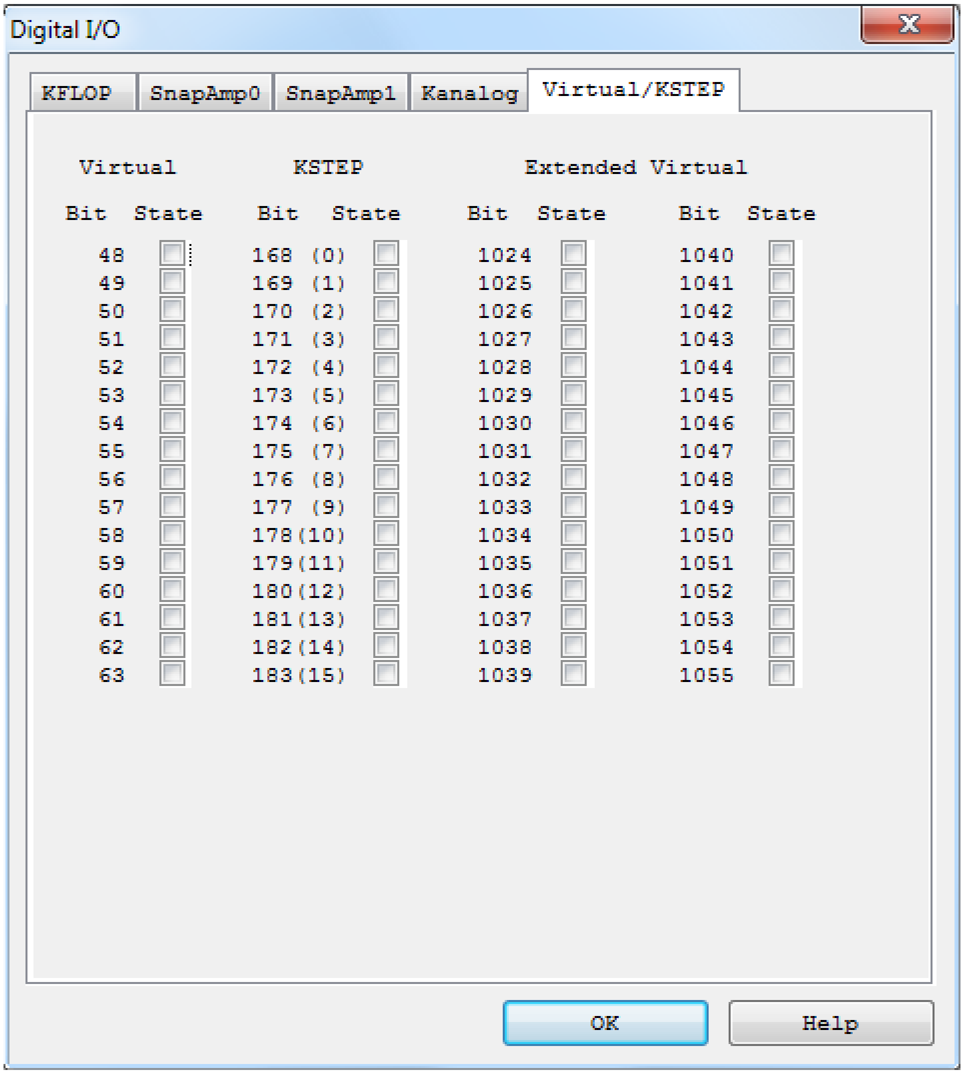
Digital I/O Screen




Digital I/O
KFLOP
SnapAmp0
SnapAmp1
Kanalog
Virtual/KSTEP
Bit
State
Output
Bit
State
64 Diff 0 Enc 0 A
65 Diff 1 Enc 0 B
66 Diff 2 Enc 1 A
67 Diff 3 Enc 1 B
68 Diff 4 Enc 2 A
69 Diff 5 Enc 2 B
70 Diff 6 Enc 3 A
71 Diff 7 Enc 3 B
72 Opto 0
73 Opto 1
74 Opto 2
75 Opto 3
76 Opto 4
77 Opto 5
78 Opto 6
79 Opto 7
80 GPIO 0
81 GPIO 1
82 GPIO 2
83 GPIO 3
84 GPIO 4
85 GPIO 5
86 GPIO 6
87 GPIO 7
88 GPIO 8
89 GPIO 9
90 GPIO 10
91 GPIO 11
92 GPIO 12
93 GPIO 13
OK
Help


Digital I/O
KFLOP
SnapAmp0
SnapAmp1
Kanalog
Virtual/KSTEP
Bit
State
Output
Bit
State
96 Diff 0 Enc 0 A
97 Diff 1 Enc 0 B
98 Diff 2 Enc 1 A
99 Diff 3 Enc 1 B
100 Diff 4 Enc 2 A
101 Diff 5 Enc 2 B
102 Diff 6 Enc 3 A
103 Diff 7 Enc 3 B
104 Opto 0
105 Opto 1
106 Opto 2
107 Opto 3
108 Opto 4
109 Opto 5
110 Opto 6
111 Opto 7
112 GPIO 0
113 GPIO 1
114 GPIO 2
115 GPIO 3
116 GPIO 4
117 GPIO 5
118 GPIO 6
119 GPIO 7
120 GPIO 8
121 GPIO 9
122 GPIO 10
123 GPIO 11
124 GPIO 12
125 GPIO 13
OK
Help


Digital I/O
KFLOP
SnapAmp0
SnapAmp1
Kanalog
Virtual/KSTEP
Bit
State
Bit
State
128 Gen Purpose IN 0
129 Gen Purpose IN 1
130 Gen Purpose IN 2
131 Gen Purpose IN 3
132 Gen Purpose IN 4
133 Gen Purpose IN 5
134 Gen Purpose IN 6
135 Gen Purpose IN 7
136 Opto IN 0
137 Opto IN 1
138 Opto IN 2
139 Opto IN 3
140 Opto IN 4
141 Opto IN 5
142 Opto IN 6
143 Opto IN 7
144 Opto OUT 0
145 Opto OUT 1
146 Opto OUT 2
147 Opto OUT 3
148 Opto OUT 4
149 Opto OUT 5
150 Opto OUT 6
151 Opto OUT 7
152 FET Driver 0
153 FET Driver 1
154 FET Driver 2
155 FET Driver 3
156 FET Driver 4
157 FET Driver 5
158 FET Driver 6
159 FET Driver 7
160 Gen Purpose 0
161 Gen Purpose 1
162 Gen Purpose 2
163 Gen Purpose 3
164 Gen Purpose 4
165 Gen Purpose 5
166 Gen Purpose 6
167 Gen Purpose 7
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.

1KFLOP 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***.

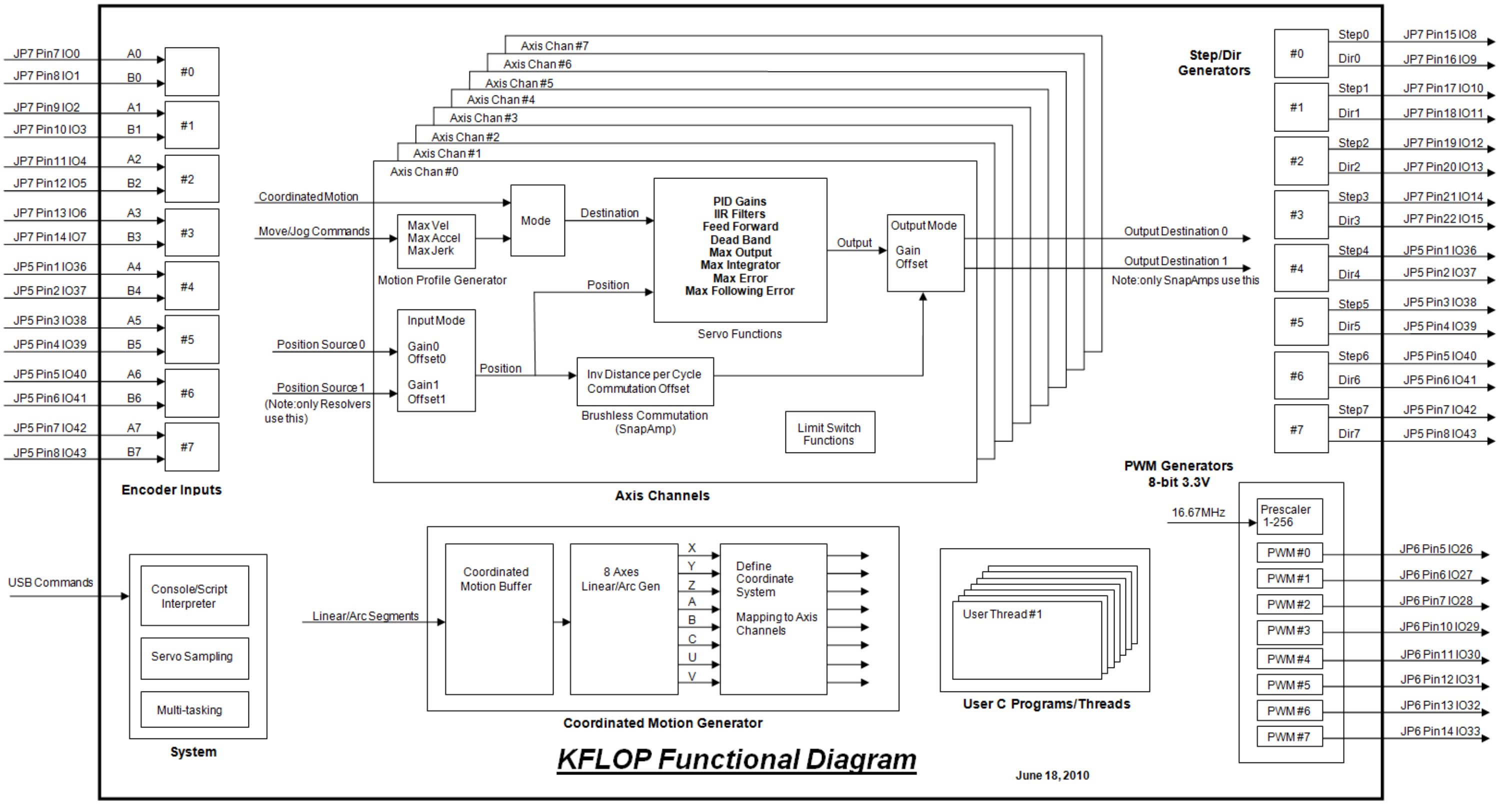
Additionally, *buffered* commands may change the state of Digital I/O bits. *Buffered* I/O commands are I/O commands that are inserted into the coordinated motion buffer. When it is required that I/O bits be changed at exact times within a motion sequence, *buffered* I/O commands may be inserted into 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 Kanalog 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. The are available for use as User status if desired.

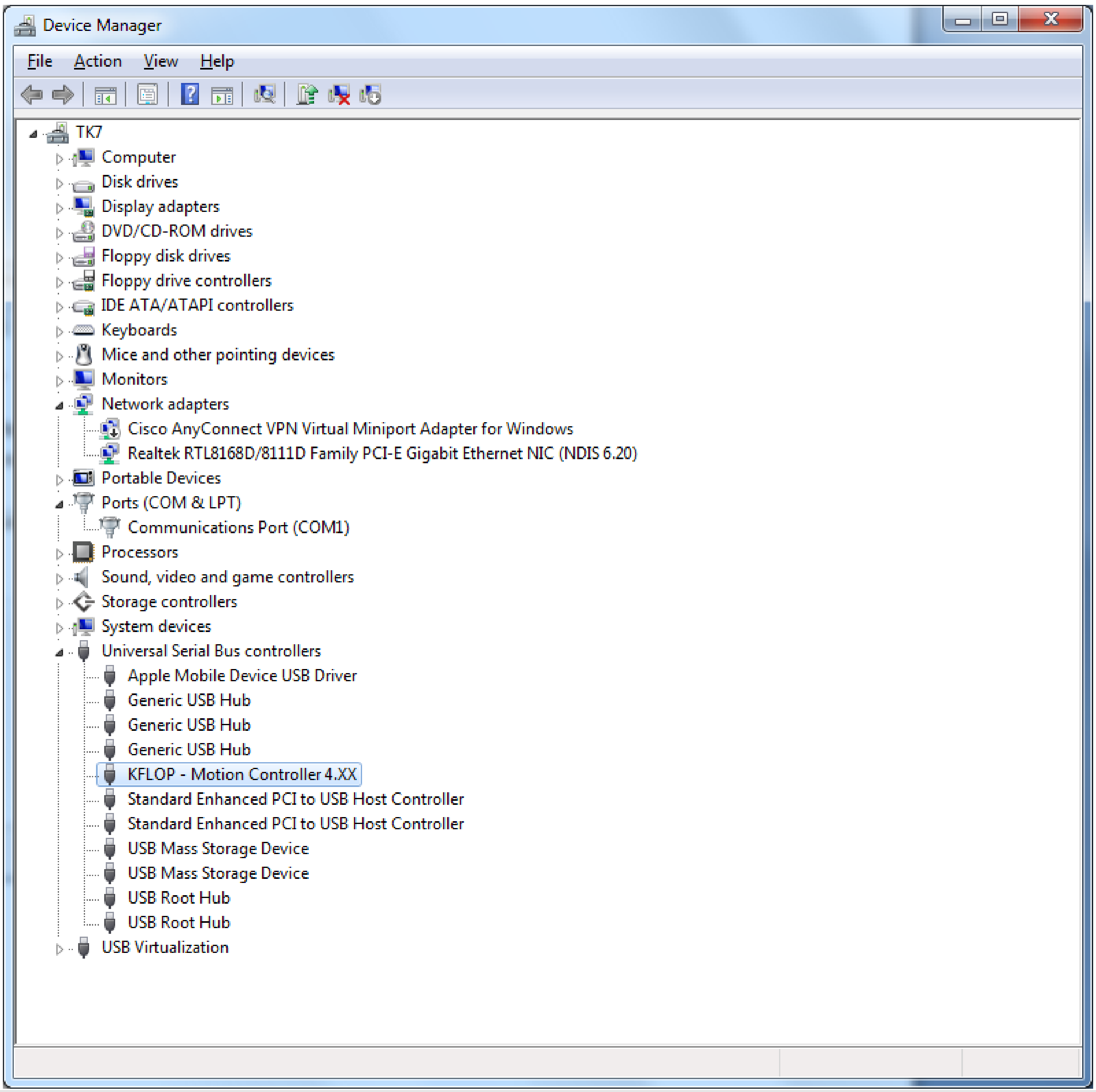
KFLOP Functional Diagram



Virtual COM Port Driver Installation

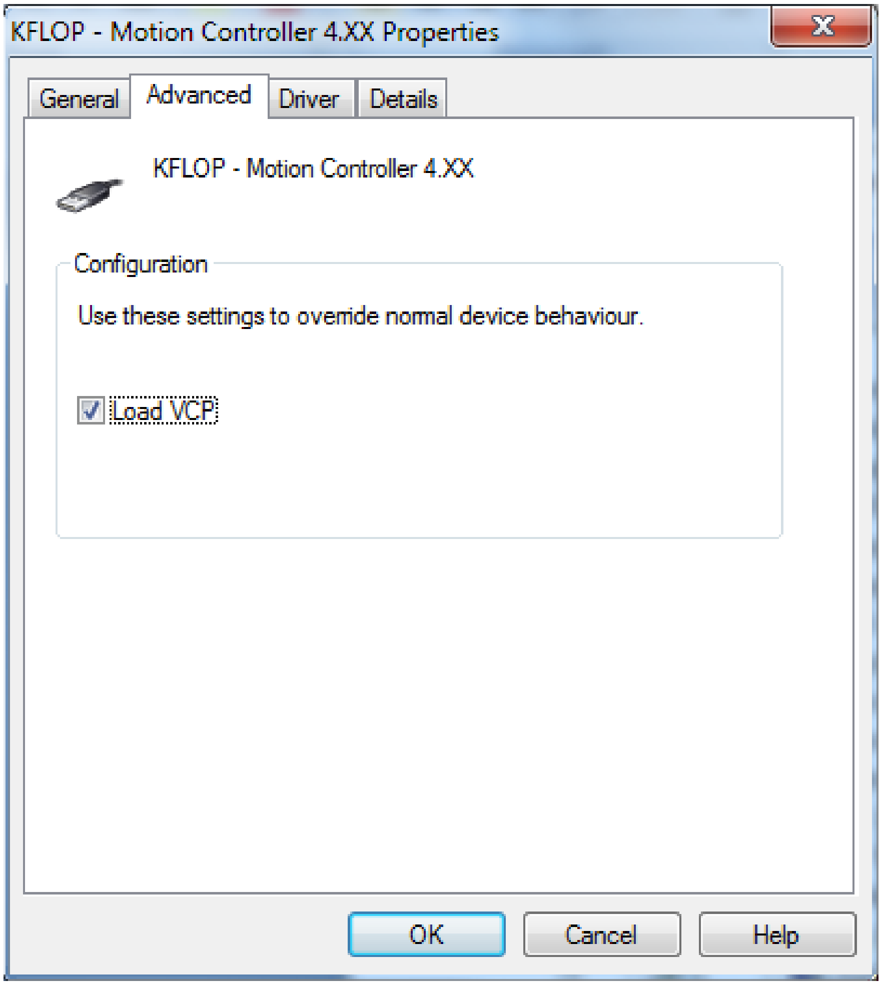
The Virtual COM Port Driver allows applications to send Script Commands to KFLOP as if it were connected as a Serial COM Port. This method is an alternate to using our much more powerful KMotion Libraries. Dynomotion's Software Applications and examples do not need to have this driver installed. It is only useful if you have some legacy Application that cannot make use of the KMotion Dynamic Linked Libraries, cannot be modified, but can communicate to a COM Port.

First install KFLOP normally using our USB Drivers. When successful there should be a KFLOP entry in Device manager as shown below:

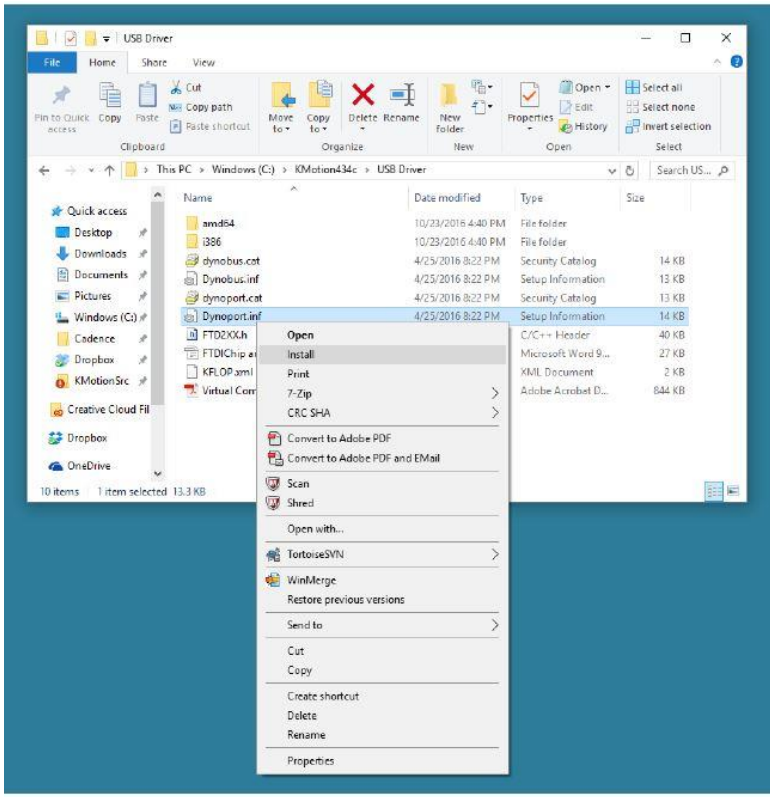


Now select “**Properties**” – “**Advanced**” – “**Load VCP**” (Virtual Com Port). Press **OK**.

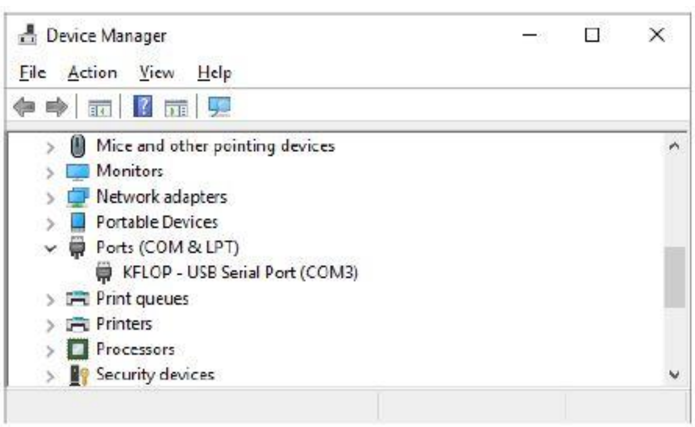
(See below if the Advanced Tab is not present)



Note: if the Advanced Tab is not present, install the Dynomotion Com Port Driver by navigating to the USB Driver Directory, right-click on Dynoport.inf, click **Install**, and **Allow** the driver to load.

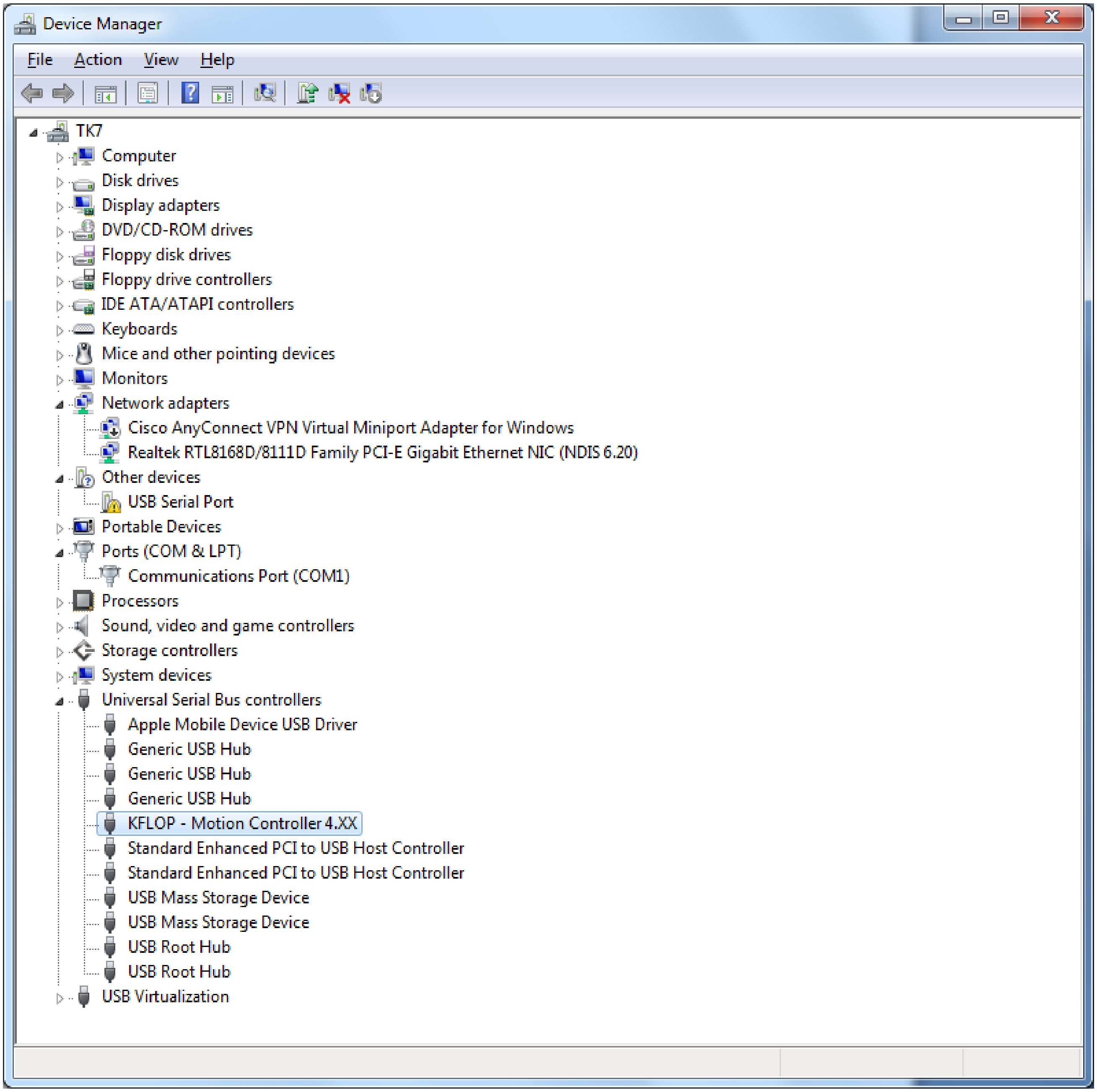


Cycling Power on KFLOP should then install the COM Port Driver completely without requiring the other steps listed below. Device Manager should now show:



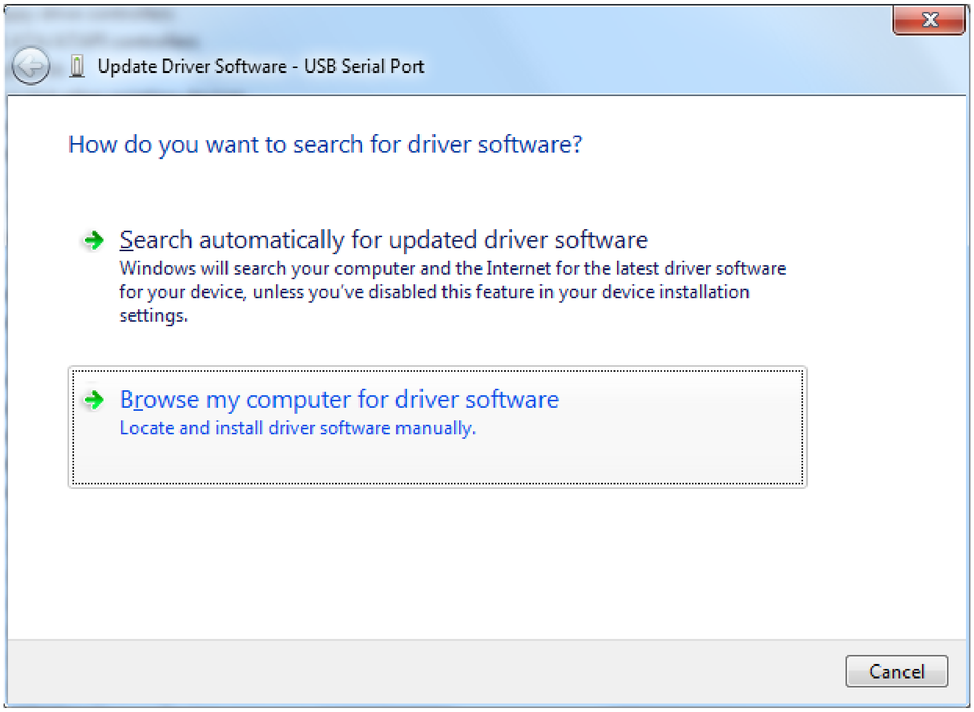
Note: if you receive an error: "The INF file you selected does not support this method of installation", you might try this alternate method using Device Manager. Delete/Rename the Dynobus.inf file so Device Manager cannot choose it.
[The INF File You selected does not support this method of installation - Windows 7 Help Forums]

Then unplug/re-plug KFLOP. There should now be a USB Serial Port – but missing the driver.

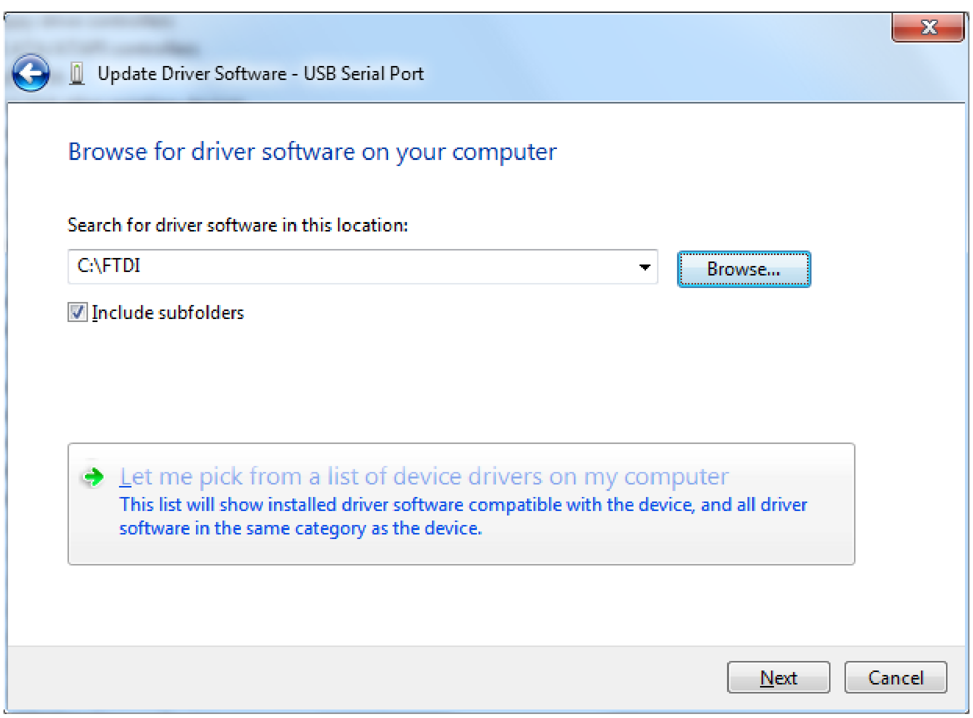


On USB Serial Port right click "**Update Driver**".

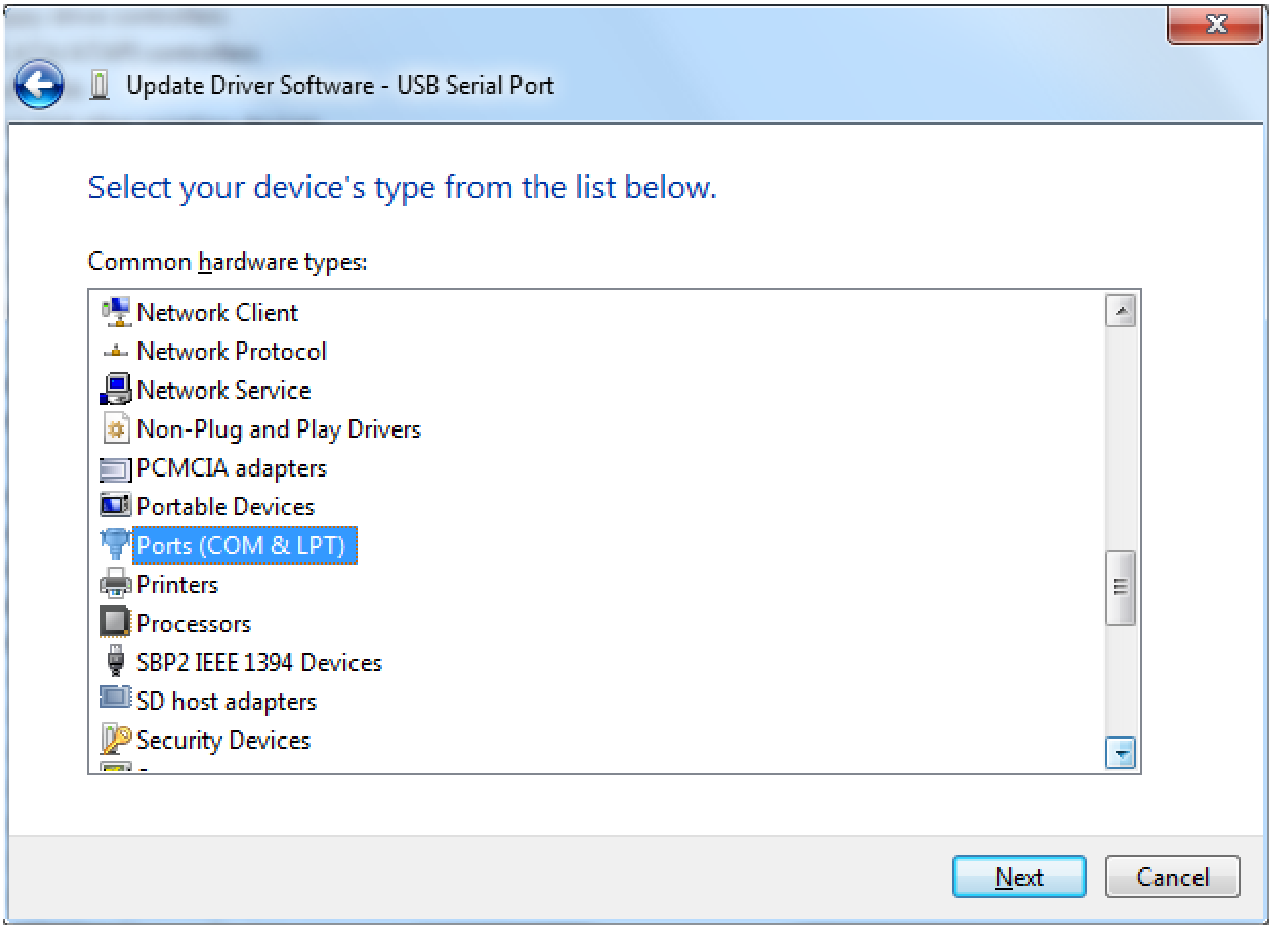
Select "**Browse my computer**".



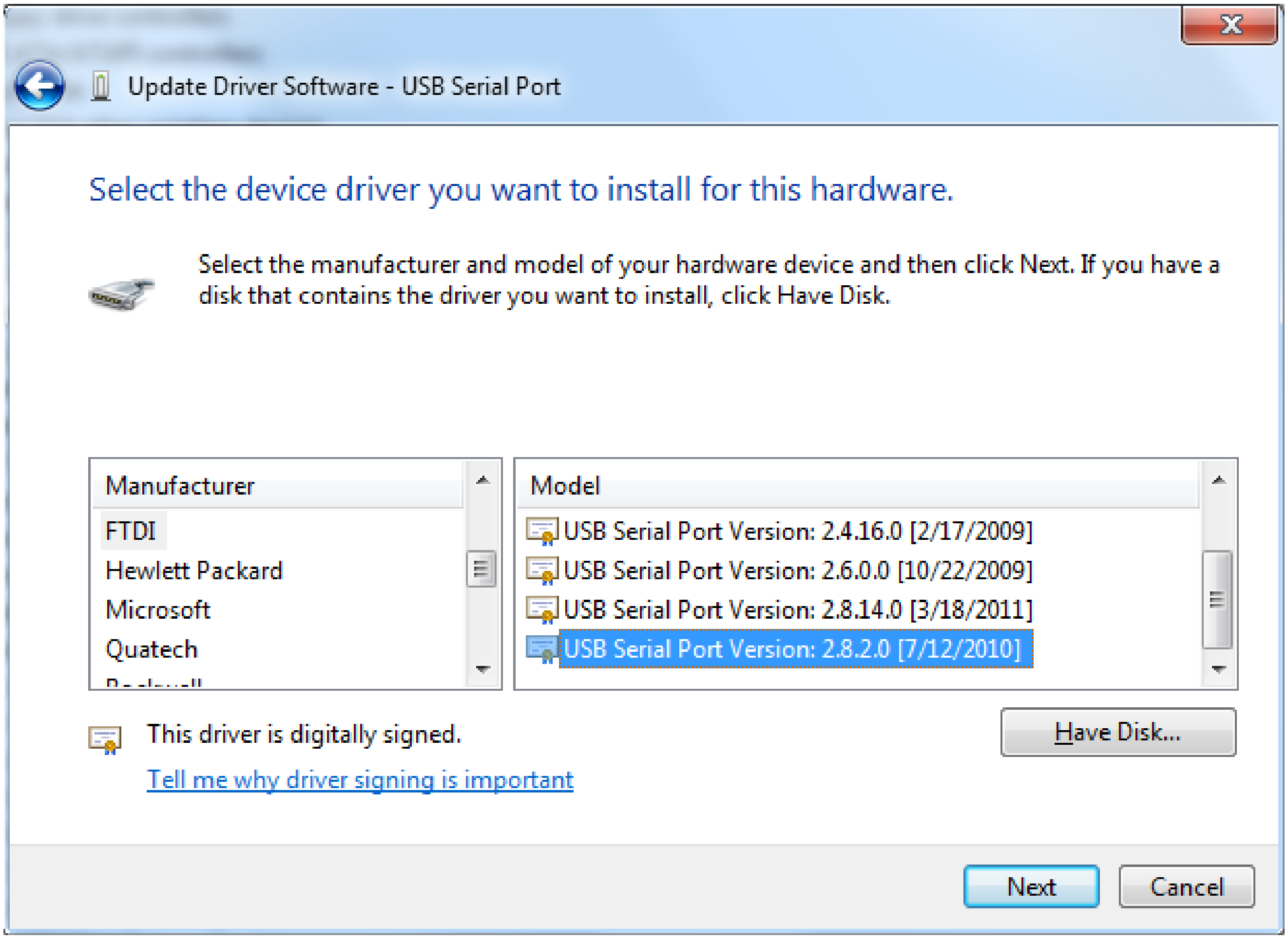
Then "**Let me pick …**"



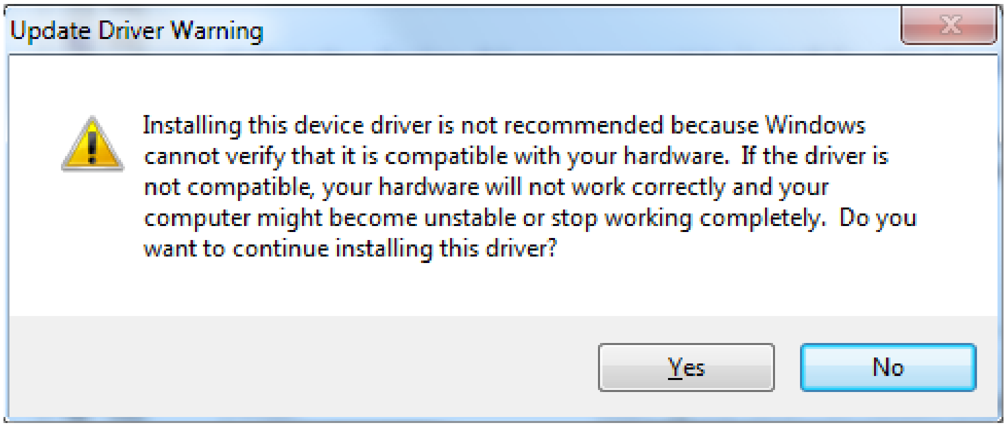
Select Device type as "**Ports**".



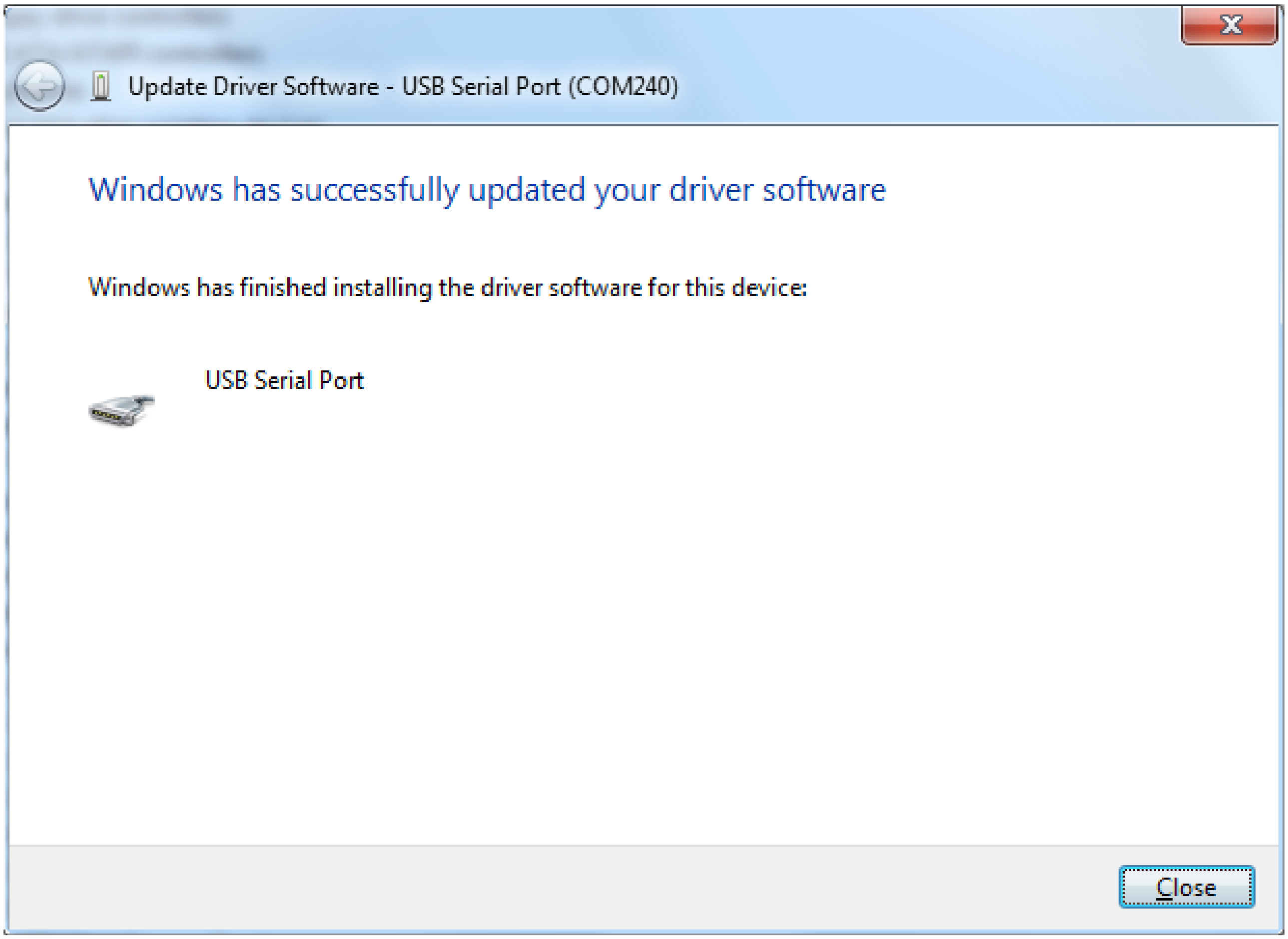
FTDI drivers should be pre-installed with Windows. Select FTDI and then latest USB Serial Port Version. Manufacturer FTDI – USB Serial Port 2.8.2.0 (or other?)



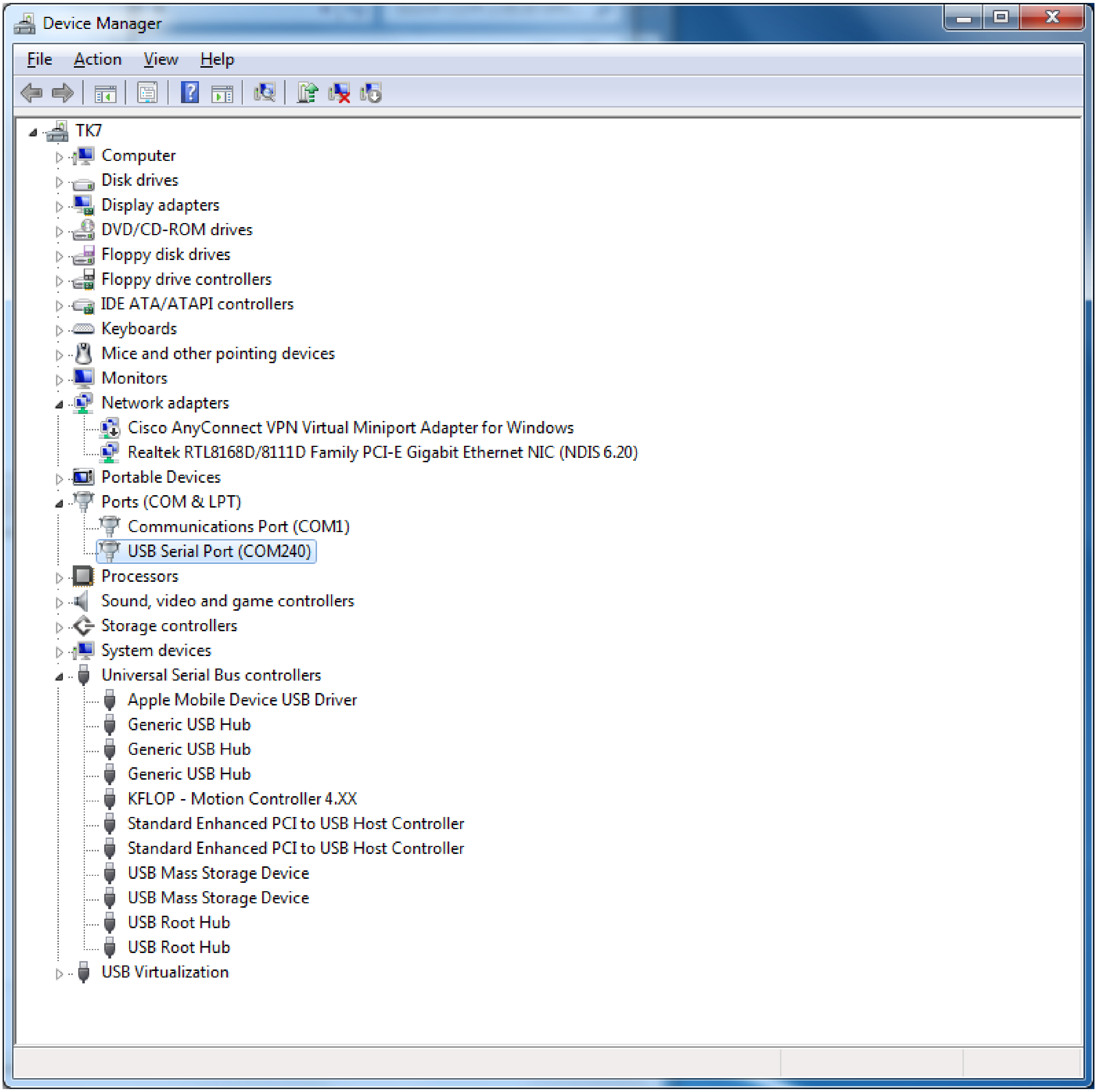
We have a different manufacturer ID, so you get this warning – choose "**Yes**".



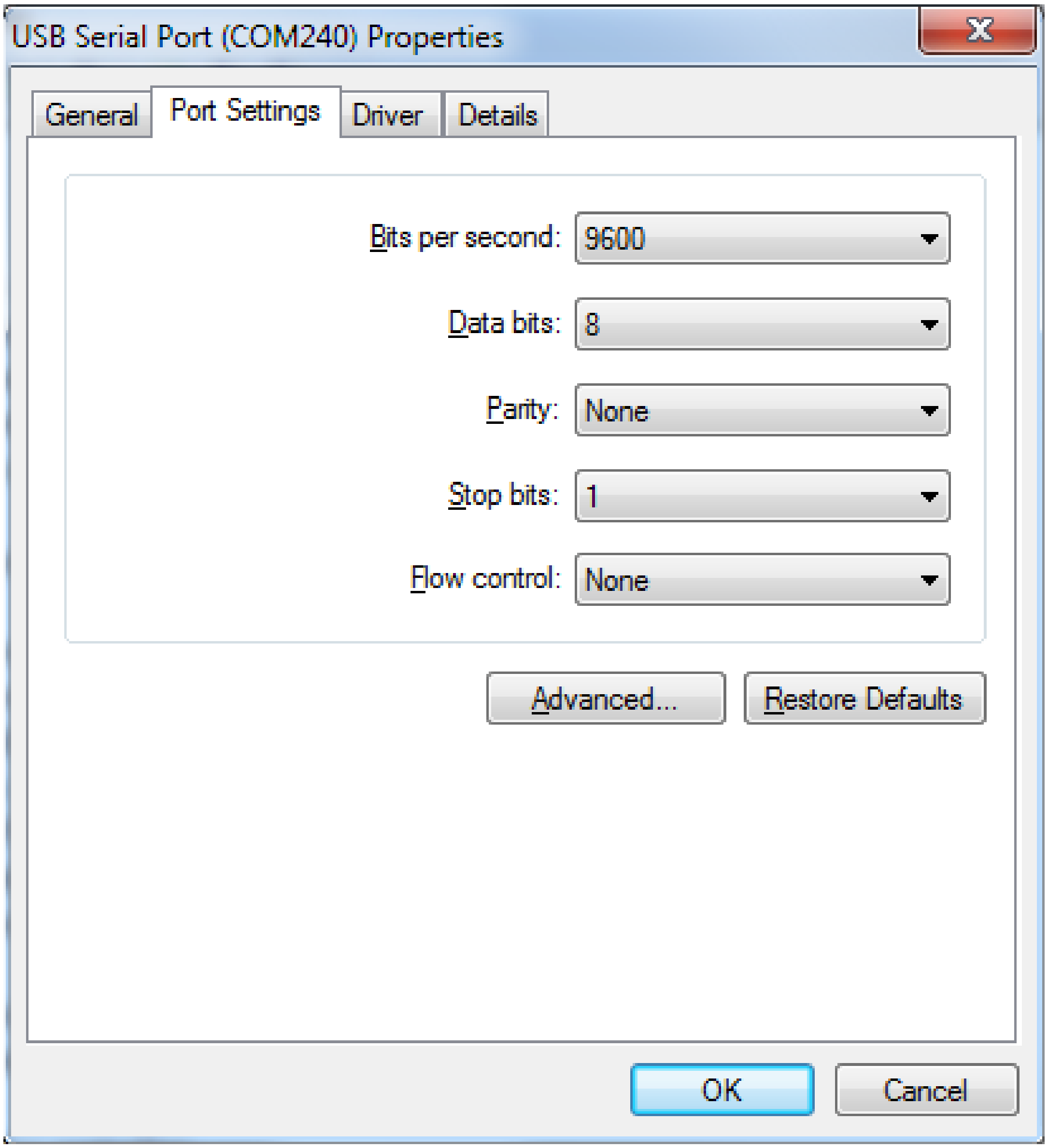
Installation should have been successful.



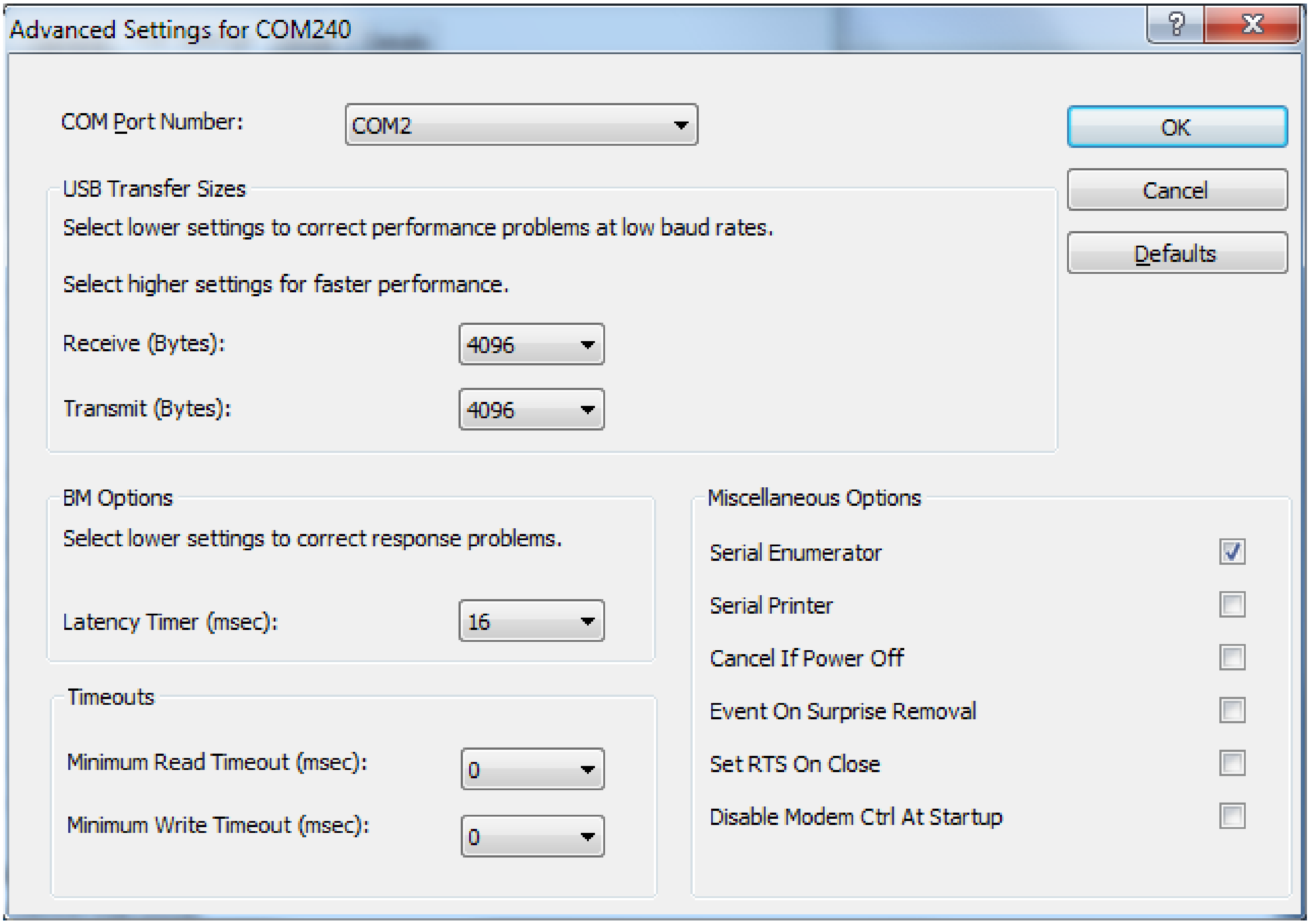
It should now work, but if you need to change the COM port number, do the following: Right click "**Properties**" – "**Port Settings**".

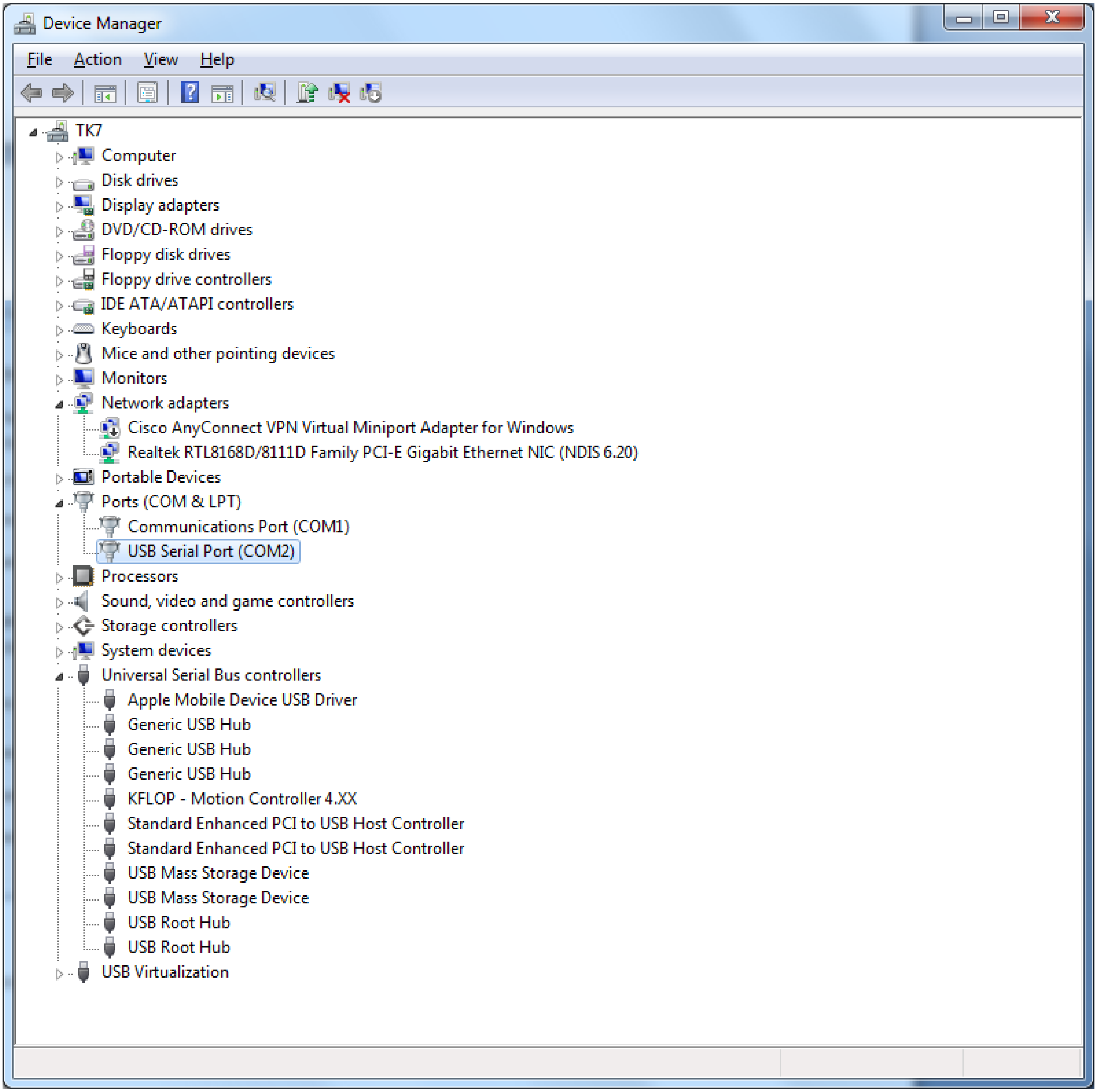


Select "**Advanced**".



Change to other COM port if necessary (COM2).

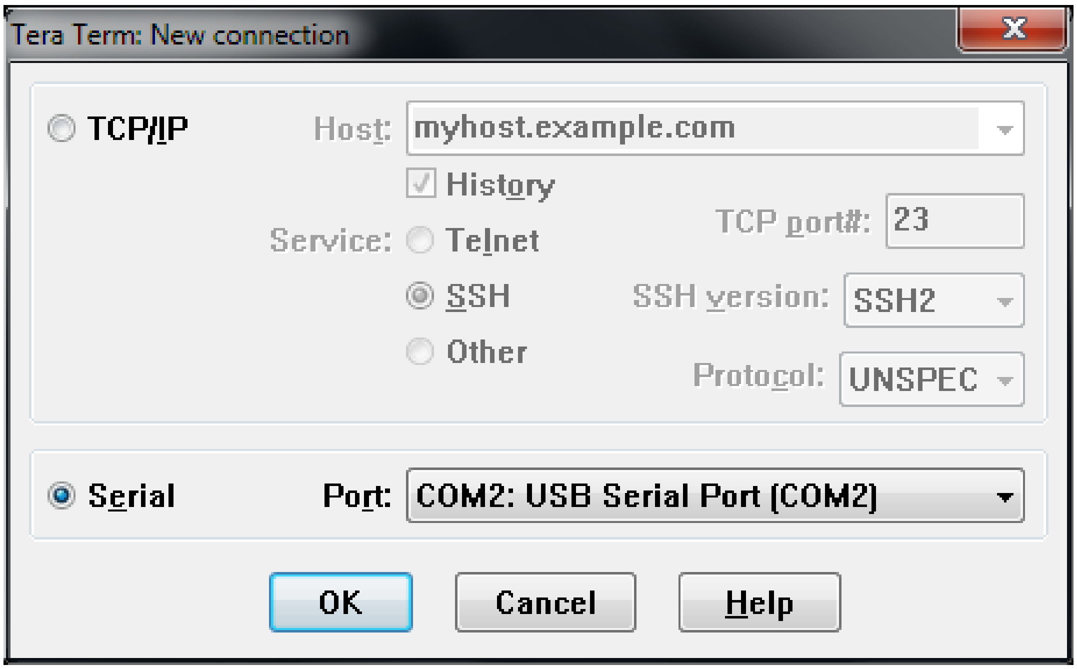



Device Manager
File Action View Help
TK7
Computer
Disk drives
Display adapters
DVD/CD-ROM drives
Floppy disk drives
Floppy drive controllers
IDE ATA/ATAPI controllers
Keyboards
Mice and other pointing devices
Monitors
Network adapters
Cisco AnyConnect VPN Virtual Miniport Adapter for Windows
Realtek RTL8168D/8111D Family PCI-E Gigabit Ethernet NIC (NDIS 6.20)
Portable Devices
Ports (COM & LPT)
Communications Port (COM1)
USB Serial Port (COM2)
Processors
Sound, video and game controllers
Storage controllers
System devices
Universal Serial Bus controllers
Apple Mobile Device USB Driver
Generic USB Hub
Generic USB Hub
Generic USB Hub
KFLOP - Motion Controller 4.XX
Standard Enhanced PCI to USB Host Controller
Standard Enhanced PCI to USB Host Controller
USB Mass Storage Device
USB Mass Storage Device
USB Root Hub
USB Root Hub
USB Virtualization

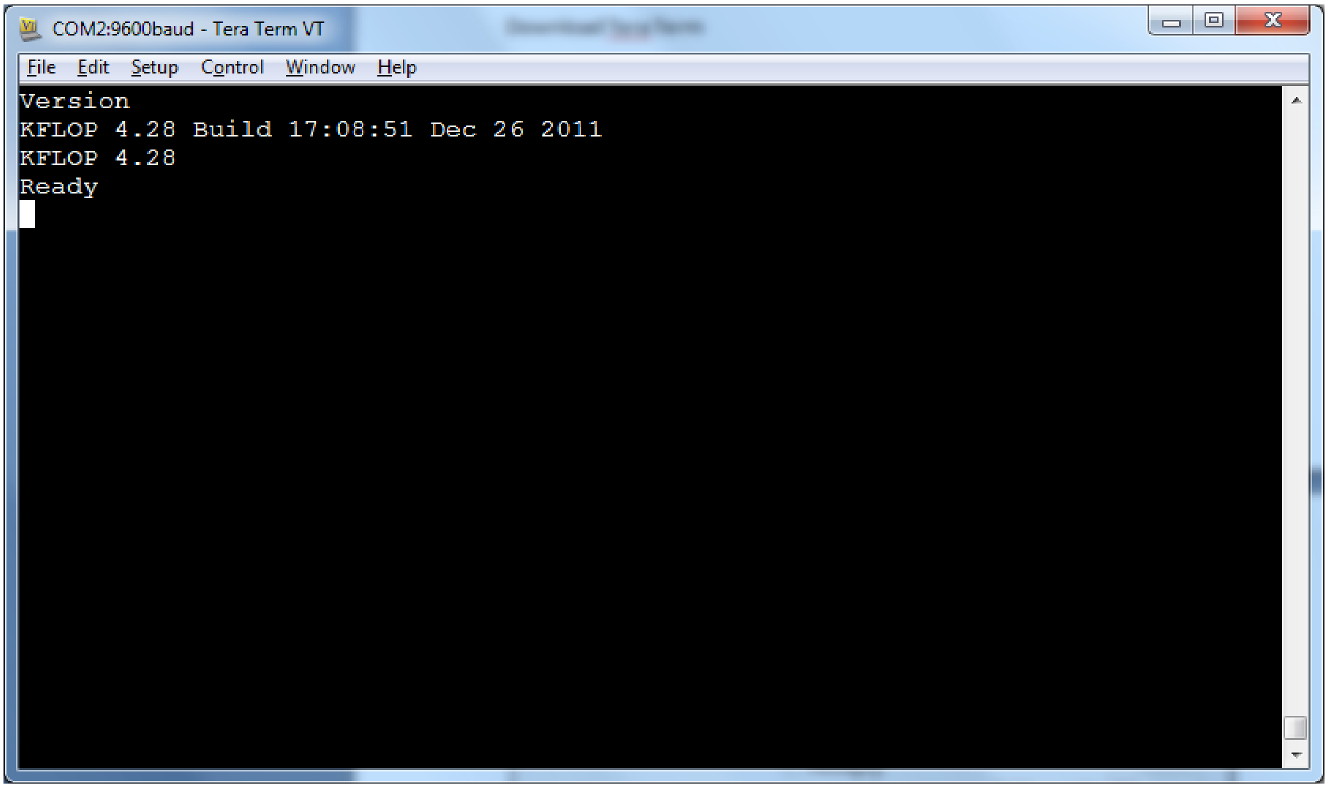
To test download and install a serial com program like Tera Term from:

http://ttssh2.sourceforge.jp

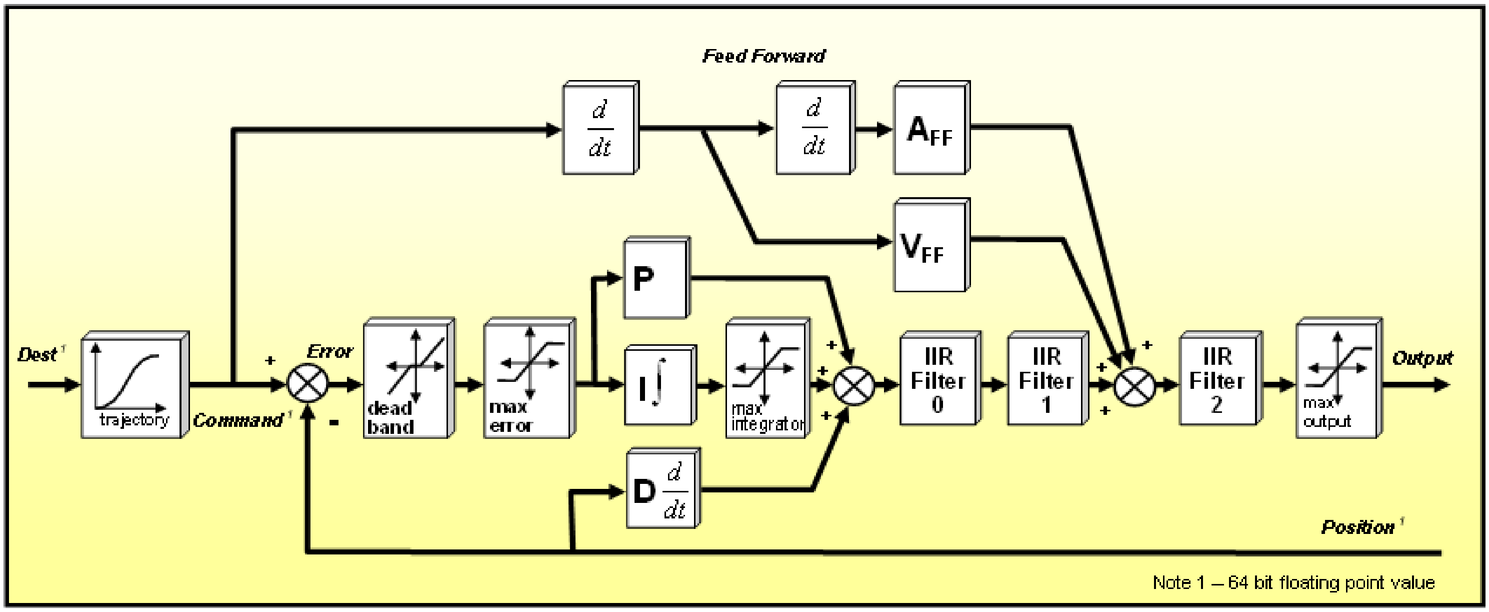
Run it and select "**Serial**" and the Com port you defined.



Type “**Version**” and KFLOP should respond.



Servo Flow Diagram



KMotion 2.2 Servo Flow Diagram

Driver Library Routines

KMotionDLL

```
CKMotionDLL(int board);
```

Creates a CKMotionDLL object to be used to communicate to a specific board.

board

specifies which board in the system the object is associated with

---

`int WriteLine(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

s

Null terminated string to send

Example

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

`int WriteLineReadLine(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

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(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

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(0);
char resp1[256];
char resp2[256];
char resp3[256];

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

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

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

// release our access to the board

KM.ReleaseToken();
```

`int WaitToken();`

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

none

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM(0);
char resp1[256];
char resp2[256];
char resp3[256];

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

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

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

// release our access to the board

KM.ReleaseToken();
```

`int KMotionLock();`

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

none

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM(0);
char resp1[256];
char resp2[256];
char resp3[256];
int result;

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

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

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

// release our access to the board

KM.ReleaseToken();
```

```
void ReleaseToken();
```

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 cannot fail

Parameters

none

specifies which board in the system the command applies to

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM(0);
char resp1[256];
char resp2[256];
char resp3[256];
int result;

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


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

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

```
// release our access to the board

KM.ReleaseToken();
```

int Failed();

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 cannot fail

Parameters

none

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM(0);

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

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

```
int LoadCoff(int Thread, const char *Name, 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.

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

Return Value

returns 0 - if successful

Parameters

Thread

KMotion Thread that the program should be loaded into

Name

Filename of coff file to download

PackToFlash

Internal system command always specify as false

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM(0);
unsigned int EntryPoint;

if (KM.LoadCoff(1, "C:\\test.out", false)) return 1;

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

`int ServiceConsole();`

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

none

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM(0);

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

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

`int SetConsoleCallback(CONSOLE_HANDLER *ch);`

Sets the user provided console callback function.

Return Value

returns 0 - if successful

Parameters

ch

name of console handler function

Example

```
#include "KMotionDLL.h"
CKMotionDLL KM(0);
int ConsoleHandler(const char *buf)
{
       MyLogData(buf);
       return 0;
}

.
.
.

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

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

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>
LimitSwitchNegBit<N>=<B>
LimitSwitchPosBit<N>=<B>
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>
SoftLimitNeg<N>=<G>
SoftLimitPos<N>=<G>
StepperAmplitude<N>=<M>
Vel<N>=<V>

I/O Commands/Status

ADC<N>
ClearBit<N>
ClearBitBuf<N>
DAC<N> <M>
GetBitDirection<N>
GetVirtualBits
ReadBit<N>
ReadDiskData<S><N>
SetBit<N>
SetBitBuf<N>
SetBitDirection<N>=<M>
SetStateBit<N>=<M>
SetStateBitBuf<N>=<M>
SPISetMode<N>=<N>
WaitBitBuf<N>
WaitNotBitBuf<N>

Output Stage

3PH<N>=<M><A>
4PH<N>=<M><A>
GetKognaPWMEne<N>=<N>
GetKognaPWMLength<N>=<N>
HRPWMSetMode<N>=<N>
PWM<N>=<M>
PWMC<N>=<M>
PWMR<N>=<M>
SetKognaPWMEne<N>=<N>
SetKognaPWMLength<N>=<N>

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>

User Threads

EntryPoint<N> <H>
CheckThread<N>
Execute<N>
Kill<N>
LoadData<H> <N>
SetStartupThread<N> <M>

FLASH Commands

ClearFlashImage
Flash
LoadFlash<H> <N>
ProgFlashImage

Misc Commands

Echo<S>
FPGA<N><M>
FPGAW<N><M>
GetAllDestVelHex
GetIpAddr
GetSerialNumber
GetPersistDec<N>
GetPersistHex<N>
GetStatus
LED
Reboot!
SetIpAddr<D><D><D><D>
SetPersistDec <O> <D>
SetPersistHex <O> <H>
USB
Version

Motion Commands

Arc<XC> <YC> <RX> <RY> <θ0> <dθ>
<Z0> <A0> <B0> <C0>
<Z1> <A1> <B1> <C1>
<a> <b> <c> <d> <tF>

ArcEx<XC> <YC> <RX> <RY> <θ0><dθ>
<Z0> <A0> <B0> <C0> <U0> <V0>
<Z1> <A1> <B1> <C1> <U1> <V1>
<a> <b> <c> <d> <tF>

ArcP<XC> <YC> <RX> <RY> <θ0> <dθ>
<Z0> <A0> <B0> <C0> <U0> <V0>
<XP0> <YP0> <ZP0> <AP0> <BP0> <CP0> <UP0> <VP0>
<Z1> <A1> <B1> <C1> <U1> <V1>
<XP1> <YP1> <ZP1> <AP1> <BP1> <CP1> <UP1> <VP1>
<a> <b> <c> <d> <tF>

ArcZX <ZC> <XC> <RX> <RY> <θ0> <dθ>
<Y0> <A0> <B0> <C0>
<Y1> <A1> <B1> <C1>
<a> <b> <c> <d> <tF>

ArcXZEx<XC> <ZC> <RX> <RY> <θ0> <dθ>
<Y0> <A0> <B0> <C0> <U0> <V0>
<Y1> <A1> <B1> <C1> <U1> <V1>
<a> <b> <c> <d> <tF>

ArcXZP<XC> <ZC> <RX> <RY> <θ0> <dθ>
<Y0> <A0> <B0> <C0> <U0> <V0>
<XP0> <YP0> <ZP0> <AP0> <BP0> <CP0> <UP0> <VP0>
<Y1> <A1> <B1> <C1> <U1> <V1>
<XP1> <YP1> <ZP1> <AP1> <BP1> <CP1> <UP1> <VP1>
<a> <b> <c> <d> <tF>

ArcYZ<YC> <ZC> <RX> <RY> <θ0> <dθ>
<X0> <A0> <B0> <C0>
<X1> <A1> <B1> <C1>
<a> <b> <c> <d> <tF>

ArcYZEx<YC> <ZC> <RX> <RY> <θ0> <dθ>
<X0> <A0> <B0> <C0> <U0> <V0>
<X1> <A1> <B1> <C1> <U1> <V1>
<a> <b> <c> <d> <tF>

ArcYZP<YC> <ZC> <RX> <RY> <θ0> <dθ>
<X0> <A0> <B0> <C0> <U0> <V0>
<XP0> <YP0> <ZP0> <AP0> <BP0> <CP0> <UP0> <VP0>
<X1> <A1> <B1> <C1> <U1> <V1>
<XP1> <YP1> <ZP1> <AP1> <BP1> <CP1> <UP1> <VP1>
<a> <b> <c> <d> <tF>

ArcXZ <XC> <ZC> <RX> <RZ>
<θ0> <dθ> <Y0> <A0> <B0> <C0>
<Y1> <A1> <B1> <C1>

<a> <b> <c> <d> <tF>

ArcYZ <YC> <ZC> <RY> <RZ>
<θ0> <dθ> <X0> <A0> <B0> <C0>
<X1> <A1> <B1> <C1>
<a> <b> <c> <d> <tF>

ArcHex <XC> <YC> <RX> <RY>
<θ0> <dθ> <Z0> <A0> <B0> <C0>
<Z1> <A1> <B1> <C1>
<a> <b> <c> <d> <tF>

ArcHexP<XC> <YC> <RX> <RY> <θ0> <dθ>
<Z0> <A0> <B0> <C0> <U0> <V0>
<XP0> <YP0> <ZP0> <AP0> <BP0> <CP0> <UP0> <VP0>
<Z1> <A1> <B1> <C1> <U1> <V1>
<XP1> <YP1> <ZP1> <AP1> <BP1> <CP1> <UP1> <VP1>
<a> <b> <c> <d> <tF>

ArcHexZX<ZC> <XC> <RX> <RY> <θ0> <dθ>
<Y0> <A0> <B0> <C0>
<Y1> <A1> <B1> <C1>
<a> <b> <c> <d> <tF>

ArcHexZXEx<ZC> <XC> <RX> <RY> <θ0> <dθ>
<Y0> <A0> <B0> <C0> <U0> <V0>
<Y1> <A1> <B1> <C1> <U1> <V1>
<a> <b> <c> <d> <tF>

ArcHexZXP<ZC> <XC> <RX> <RY> <θ0> <dθ>
<Y0> <A0> <B0> <C0> <U0> <V0>
<XP0> <YP0> <ZP0> <AP0> <BP0> <CP0> <UP0> <VP0>
<Y1> <A1> <B1> <C1> <U1> <V1>
<XP1> <YP1> <ZP1> <AP1> <BP1> <CP1> <UP1> <VP1>
<a> <b> <c> <d> <tF>

ArcHexXZ<XC> <ZC> <RX> <RZ>
<θ0> <dθ> <Y0> <A0> <B0> <C0>
<Y1> <A1> <B1> <C1>
<a> <b> <c> <d> <tF>

ArcHexYZ <YC> <ZC> <RY> <RZ>
<θ0> <dθ> <X0> <A0> <B0> <C0>
<X1> <A1> <B1> <C1>
<a> <b> <c> <d> <tF>

ArcHexYZEx<YC> <ZC> <RX> <RY> <θ0> <dθ>
<X0> <A0> <B0> <C0> <U0> <V0>
<X1> <A1> <B1> <C1> <U1> <V1>
<a> <b> <c> <d> <tF>

ArcHexYZP<YC> <ZC> <RX> <RY> <θ0> <dθ>
<X0> <A0> <B0> <C0> <U0> <V0>
<XP0> <YP0> <ZP0> <AP0> <BP0> <CP0> <UP0> <VP0>
<X1> <A1> <B1> <C1> <U1> <V1>
<XP1> <YP1> <ZP1> <AP1> <BP1> <CP1> <UP1> <VP1>
<a> <b> <c> <d> <tF>

BegRapidBuf
CheckDone<N>
CheckDoneBuf
CheckDoneXYZABC
ConfigSpindle <T> <A> <U> <W> <C>
DefineCS <X> <Y> <Z> <A><B><C>
DefineCSEX <X> <Y> <Z> <A> <B> <C> <U> <V>
DisableAxis<N>
EndRapidBuf
EnableAxis<N>
EnableAxisDest<N> <M>
Enabled<N>
ExecBuf
ExecTime
FlushBuf
GetSpindleRPS
GetStopState
Jog<N>=<V>
JogAtAccel<N>=<V><A>

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

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

LinearP <X0> <Y0> <Z0> <A0> <B0> <C0> <U0> <V0>
<XP0> <YP0> <ZP0> <AP0> <BP0> <CP0> <UP0> <VP0>
<X0> <Y0> <Z0> <A0> <B0> <C0> <U0> <V0>
<XP1> <YP1> <ZP1> <AP1> <BP1> <CP1> <UP1> <VP1>
<a> <b> <c> <d> <tF>

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

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

LinearHexP <X0> <Y0> <Z0><A0> <B0> <C0> <U0><V0>
<XP0> <YP0> <ZP0> <AP0> <BP0> <CP0> <UP0> <VP0>
<X1> <Y1> <Z1> <A1> <B1> <C1> <U1> <V1>
<XP1> <YP1> <ZP1> <AP1> <BP1> <CP1> <UP1> <VP1>
<a> <b> <c> <d> <tF>

LHex1 <X1> <Y1> <Z1> <A1> <B1> <C1>
<a> <b> <c> <d> <tF>

LHexEx1<X1> <Y1> <Z1> <A1> <B1> <C1> <U1> <V1>
<a> <b> <c> <d> <tF>

LHexP1<X1> <Y1> <Z1> <A1><B1> <C1> <U1><V1>
<XP1> <YP1> <ZP1> <AP1> <BP1> <CP1> <UP1> <VP1>
<a> <b> <c> <d> <tF>

LHex2 <a> <b> <c> <d> <tF>

LHexP2 <a> <b> <c> <d> <tF>

Move<N>=<M>
MoveAtVel<N>=<M><V>
MoveAtVelAccel<N>=<D> <V>
MoveExp<N>=<D> <T>
MoveRel<N>=<M>
MoveRelAtVel<N>=<M><V>
MoveRelAtVelAccel<N>=<D><V><A>
MoveRelAtVelAccelSoft<N>=<D><V><A>
MoveXYZABC<X><Y><Z> <A><B><C>
OpenBuf
SetFRO <F>
SetFROTemp <F>
SetFROwRate <F> <R>
SetFROwRateTemp <F> <R>
SetRapidFRO <F>
SetRapidFROwRate <F> <R>
StopImmediate<M>
TrigThread <S>
Zero<N>

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 sec^2

Example

Accel0=1000.0

ADC<*N*>

Description

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

Channels 0-3 are ±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> <$d\theta$> <Z_0> <A_0> <B_0> <C_0> <Z_1> <A_1> <B_1> <C_1> <a> <b> <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

$$p = \mathbf{a}\, t^3 + \mathbf{b}\, t^2 + \mathbf{c}\, t + \mathbf{d}$$

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 $\boldsymbol{t_F}$, where ***t*** varies from **0** ... $\boldsymbol{t_F}$. 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 phase over the path. ie: ***t*** might vary from 0.0->0.1 where p might vary from 0.3->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

<θ_0> - initial theta position on ellipse, radians (0 radians points in the +*x* direction)

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

<Z_0> - initial Z position on path, units are position units of z axis

<A_0> - initial A position on path, units are position units of a axis

<B_0> - initial B position on path, units are position units of b axis

<C_0> - initial C position on path, units are position units of c axis

<Z_1> - final Z position on path, units are position units of z axis

<A_1> - final A position on path, units are position units of a axis

<B_1> - final B position on path, units are position units of b axis

<*C*$_1$> - final C position on path, units are position units of c axis

<*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 (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

ArcEx *<XC> <YC> <RX> <RY> <θ0> <dθ> <Z0> <A0> <B0> <C0> <U0> <V0> <Z1> <A1> <B1> <C1> <U1> <V1> <a> <b> <c> <d> <tF>*

Description

Place circular (also elliptical or helical) interpolated move into the coordinated motion buffer. Same as Arc command except expands the axes from 6 to 8 to include U V axes.

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

ArcEx 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 0.0 0.0 0.0 0.0 0.1 0.0 10.0

ArcP *<XC> <YC> <RX> <RY> <θ0> <dθ> <Z0> <A0> <B0> <C0> <U0> <V0> <XP0> <YP0> <Z1> <A1> <B1> <C1> <U1> <V1> <XP1> <YP1> <ZP1> <AP1> <BP1> <CP1> <UP1> <VP1> <a> <b> <c> <d> <tF>* (Kogna only)

Description

Place circular (also elliptical or helical) interpolated move into the coordinated motion buffer. Same as ArCEx command except expands the axes from 8 to 16 to include the 8 prime axes XP YP ZP AP BP CP UP VP axes.

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

ArcP 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 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 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

ArcZX*<ZC> <XC> <RX> <RZ> <θ0> <dθ> <Y0> <A0> <B0> <C0> <Y1> <A1> <B1> <C1> <a> <b> <c> <d><tF>*

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 ZX plane rather than the XY plane.

ArcXZEx*<XC> <ZC> <RX> <RY> <θ0> <dθ> <Y0> <A0> <B0> <C0> <U0> <V0> <Y1> <A1> <B1> <C1> <U1> <V1> <a> <b> <c> <d> <tF>*

Description

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

ArcXZP*<XC> <ZC> <RX> <RY> <θ0> <dθ> <Y0> <A0> <B0> <C0> <U0> <V0> <XP0> <YP0> <Y1> <A1> <B1> <C1> <U1> <V1> <XP1> <YP1> <ZP1> <AP1> <BP1> <CP1> <UP1> <VP1> <a> <b> <c> <d> <tF>* (Kogna only)

Description

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

ArcYZ*<YC> <ZC> <RY> <RZ> <θ0> <dθ> <X0> <A0> <B0> <C0> <X1> <A1> <B1> <C1> <a> <b> <c><d> <tF>*

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.

ArcYZEx*<YC> <ZC> <RX> <RY> <θ0> <dθ> <X0> <A0> <B0> <C0> <U0> <V0> <X1> <A1> <B1> <C1> <U1> <V1> <a> <b> <c> <d> <tF>*

Description

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

ArcYZP*<YC> <ZC> <RX> <RY> <θ0> <dθ> <X0> <A0> <B0> <C0> <U0> <V0> <XP0> <YP0> <X1> <A1> <B1> <C1> <U1> <V1> <XP1> <YP1> <ZP1> <AP1> <BP1> <CP1> <UP1> <VP1> <a> <b> <c> <d> <tF>* (Kogna only)

Description

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

Arc*Hex <X_C> <Y_C> <R_X> <R_Y> <θ_0> <dθ> <Z_0> <A_0> <B_0> <C_0> <Z_1> <A_1> <B_1> <C_1> <a> <b> <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 0 40c8f5c3 0 41800000

ArcHexEx*<XC> <YC> <RX> <RY> <θ0> <dθ> <Z0> <A0> <B0> <C0> <U0> <V0> <Z1> <A1> <B1> <C1> <U1> <V1> <a> <b> <c><d> <tF>*

Description

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

Parameters

See above.

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

ArcHex 3f000000 3f000000 3f800000 3f800000 0 40c90fdb 0 0 0 0 0 0 0 0 0 0 0 0 0 3dcccccd 0 41200000

ArcHexP*<XC> <YC> <RX> <RY> <θ0> <dθ> <Z0> <A0> <B0> <C0> <Z1> <A1> <B1> <C1> <a> <b> <c><d> <tF>* (Kogna only)

Description

Place circular (also elliptical or helical) interpolated move into the coordinated motion buffer. This command is exactly the same as the ArcP command above, except all 39 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 A B C motion, performed in 10 seconds)

ArcHexP 3f000000 3f000000 3f800000 3f800000 0 40c90fdb 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3dcccccd 0 41200000

ArcHexZX*<ZC> <XC> <RX> <RY> <θ0> <dθ> <Y0> <A0> <B0> <C0> <U0> <V0> <Y1> <A1> <B1> <C1> <U1> <V1> <a> <b> <c> <d> <tF>*

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 ZX plane rather than the XY plane.

ArcHexZXEx*<ZC> <XC> <RX> <RZ> <θ0> <dθ> <Y0> <A0> <B0> <C0> <U0> <V0> <Y1> <A0> <B0> <C0> <U1> <V1> <a> <b> <c> <d> <tF>*

Description

Place circular (also elliptical or helical) interpolated move into the coordinated motion buffer. Same as ArcHexEx Command except circular motion is performed in the ZX plane rather than the XY plane.

ArcHexZXP*<ZC> <XC> <RX> <RY> <θ0> <dθ> <Y0> <A0> <B0> <C0> <U0> <V0> <XP0> <YP0> <Y1> <A1> <B1> <C1> <U1> <V1> <XP1> <YP1> <ZP1> <AP1> <BP1> <CP1> <UP1> <VP1> <a> <b> <c> <d> <tF>* (Kogna only)

Description

Place circular (also elliptical or helical) interpolated move into the coordinated motion buffer. Same as ArcHexP Command except circular motion is performed in the ZX plane rather than the XY plane.

ArcHexYZ*<YC> <ZC> <RY> <RZ> <θ0> <dθ> <X0> <A0> <B0> <C0> <X1> <A1> <B1> <C1> <a> <b> <c> <d> <tF>*

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.

ArcHexYZEx*<YC> <ZC> <RX> <RY> <θ0> <dθ> <X0> <A0> <B0> <C0> <U0> <V0> <X1> <A1> <B1> <C1> <U1> <V1> <a> <b> <c> <d> <tF>*

Description

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

ArcHexYZP*<YC> <ZC> <RX> <RY> <θ0> <dθ> <X0> <A0> <B0> <C0> <U0> <V0> <XP0> <YP0> <X1> <A1> <B1> <C1> <U1> <V1> <XP1> <YP1> <ZP1> <AP1> <BP1> <CP1> <UP1> <VP1> <a> <b> <c> <d> <tF>* (Kogna only)

Description

Place circular (also elliptical or helical) interpolated move into the coordinated motion buffer. Same as ArcHexP 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

BegRapidBuf

Description

Inserts into coordinated move buffer a command to indicate Rapid is in progress and to use Rapid FRO.

Parameters

None

Example

BegRapidBuf

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 Bode 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

CheckThread<*N*>

Description

Checks whether a User Program Thread is currently executing. Returns 1 if executing, 0 if not executing.

Parameters

<N>

Thread number specified as a decimal number. Valid range 1...7

Example

CheckThread0

ClearBit<*N*>

Description

Clears an actual I/O bit or virtual I/O bit. Note that actual I/O bits must be previously defined as an output, see SetBitDirection

Parameters

<N>

Bit number specified as a decimal number. Accepted range 0...2047

Example

ClearBit0

ClearBitBuf<*N*>

Description

Inserts into the coordinated move buffer a command to clear an I/O bit (actual IO bits must be defined as an output, see SetBitDirection)

Parameters

<N>

Bit Number to clear. Accepted range 0...2047

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.

PhaseA = OutputMagnitude * sin((Position+CommutationOffset)*invDistPerCycle*2π)

PhaseB = OutputMagnitude * sin((Position+CommutationOffset)*invDistPerCycle*2π + 2π/3)

PhaseC = OutputMagnitude * sin((Position+CommutationOffset)*invDistPerCycle*2π + 4π/3)

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

PhaseA = OutputMagnitude * sin((Position+CommutationOffset)*invDistPerCycle*2π)

PhaseB = OutputMagnitude * cos((Position+CommutationOffset)*invDistPerCycle*2π)

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*> <*B*> <*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.

<B>

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

DefineCSEX*<X> <Y> <Z> <A> <B> <C> <U> <V>*

or

DefineCSEX

Description

Set or get the defined X Y Z A B C coordinate system axis assignments for up to 8 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.

<B>

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.

<U>

Assigned Axis channel number for *U*. Valid range -1 ... 7.
Use -1 if axis is not defined.

<V>

Assigned Axis channel number for *V*. Valid range -1 ... 7.
Use -1 if axis is not defined.

Example

DefineCSEX

DefineCSEX = 0 1 2 3 4 5 6 7

DefineCSP =*<XP><YP><ZP><AP><BP><CP><UP><VP>* (Kogna only)

Description

Define the 8 Axes that make up the XP YP ZP AP BP CP UP VP Prime Coordinate System. Set unused Axes to -1.

Parameters

Selected Axis for command. Valid range 0...7 (KFLOP) 0 ...15 (Kogna).

Example

DefineCSP = 0 1 -1 -1 -1 -1 -1 -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

EndRapidBuf

Description

Inserts into coordinated move buffer a command to indicate Rapid has been completed and to no longer use Rapid FRO.

Parameters

None

Example

EndRapidBuf

EntryPoint<*N*> <*H*>

Description

Set execution start address of user thread to specified address. This operation if 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 sec^2.

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

FlushBuf

Description

Informs KFLOP that the Coordinated Motion Buffer has been Flushed. This permits KFLOP to execute to the end of the buffer without performing protection against buffer starvation which would normally perform Feed Rate reduction in an attempt to avoid buffer underflow.

Parameters

None

Example

FlushBuf

FPGA<*N*> <*M*>

Description

Directly write an 8-bit value to an FPGA register. Should be only used with caution.

Parameters

<N>

FPGA Register address to write as a decimal number. Valid range 0...1023.

<M>

8-bit value as a decimal number. Valid range 0...255.

Example

FPGA 261 192

FPGAW<*N*> <*M*>

Description

Directly write a 16-bit value to an FPGA register. Should be only used with caution.

Parameters

<N>

FPGA Register address to write as a decimal number. Valid range 0...1023.

<M>

16-bit value as a decimal number. Valid range 0...65536.

Example

FPGAW 5 263

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

GetAllDestHex

Description

Get all 8 Axis Destinations as 64 bit doubles, each as 2 32-bit Hexadecimal Values (low|high)

Parameters

None

Example
Axis 0 Destination of 1000.0
GetAllDestHex
00000000 408F4000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

GatherStep0 1000.0 2000

GetAllDestVelHex

Description

Get all 8 Axis Destinations and Velocities as 64 bit doubles, each as 2 32-bit Hexadecimal Values (low|high)

Parameters

None

Example

GetAllDestVelHex

GetBitDirection<*N*>

Description

Displays whether an IO bit N (0…2047) is defined as input (0) or output (1)

Parameters

<N>

I/O bit number. Accepted range 0...2047

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

GetIPAddr (Kogna only)

Description

Get board's Ethernet IP Address.

Parameters

None

Example

GetIpAddr

GetKognaPWMEne<*N*>=<*N*> (Kogna only)

Description

Get Kogna PWM. Enable channels 0-7. 1=Enable 0=Disable.

Parameters

<N>

PWM channel number. Valid range 0...7.

Example

GetKognaPWMEn0

GetKognaPWMLength<*N*>=<*N*> (Kogna only)

Description

Get Kogna PWM Pulse Length channels 0-7. 0-255 counts.

Parameters

<N>

PWM channel number. Valid range 0...7.

Example

GetKognaPWMLength0

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

GetSerialNumber (Kogna only)

Description

Get board's Serial Number.

Parameters

None

Example

GetSerialNumber

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 stucture is uploaded as a binary image represented as 32-bit hexadecimal values.

Parameters

None

Example

GetStatus

GetStopState

Description

Reports the state of any feedhold stop in progress. 0 = not stopping, 1=stopping a coord motion, 2=stopping an independent motion of one or more axes, 3=fully stopped, 4=independent motion of all related axes fully stopped. This returns the KFLOP C program variable - CS0_StoppingState. A feedhold stop can be initiated from C code or from the Console Command StopImmediate.

Parameters

None

Example

GetStopState

GetVirtualBits

Description

KFLOP supports an extended range of 1024 virtual I/O bits. These bits reside in KFLOP's memory as a table of 32 32-bit words. The command GetVirtualBits is used to upload KFLOP virtual bit words to the PC in bulk. Any consecutive group of words can be uploaded with a single command. The words are uploaded as hexadecimal values.

Parameters

Starting word in decimal
Number of words in decimal

Example

If the top 4 bits of virtual bit word #1 (the second word, bits 1084, 1085, 1086, 1087) are set with all other bits zero, then the command shown below to display 3 words starting at word #1 would display:

GetVirtualBits 1 3
F0000000 0 0

HRPWMSetMode<*N*>=<*M*> (Kogna only)

Description

Set HRPWM Pin Mode mux channels 0-3 1=GPIO 0=HRPWM

Parameters

<N>

HRPWM channel number. Valid range 0...3.

<M>

Mode. Valid range 0...1. 1=GPIO, 0=HRPWM.

Example

HRPWMSetMode0=1

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>

Cuttoff 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>

*C*hannel 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 (from PC-DSP.h):

```
#define NO_INPUT_MODE 0
#define ENCODER_MODE 1
#define ADC_MODE 2
#define RESOLVER_MODE 3
#define 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 of 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 sec^3

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

JogAtAccel<*N*>=<*V*> <A>

Description

Move Axis N at Velocity V using specified Acceleration. Uses Jerk parameter for the axis. Specify zero Velocity to decelerate to stop.

Parameters

<N>

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

<V>

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

<A>

Specified Acceleration.

Example

JogAtAccel0=-200.5 5000.0

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

LED<*N*>=<*B*>

Description

Turn LED number 0 or 1 on or off.

Parameters

<N>

0 or 1 for the two LEDs.

<B>

1 = on, 0 = off.

Example

LED=1=1

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

LimitSwitchNegBit*<N>*=*<H>*

Description

Configure Limit Switch Negative Bit for Axis. Specify Decimal Value.

See also Configuration Screen.

Parameters

<N>

Axis channel number. Valid range 0...7 (KFLOP) 0...15 (Kogna).

<B>

Bit number. Vaild range 0...2047.

Example

LimitSwitchNegBit2 1024

LimitSwitchPosBit*<N>*=*<H>*

Description

Configure Limit Switch Positive Bit for Axis. Specify Decimal Value.

See also Configuration Screen.

Parameters

<N>

Axis channel number. Valid range 0...7 (KFLOP) 0...15 (Kogna).

<B>

Bit number. Vaild range 0...2047.

Example

LimitSwitchNegBit2 1024

Linear <X_0> <Y_0> <Z_0> <A_0> <B_0> <C_0> <X_1> <Y_1> <Z_1> <A_1> <B_1> <C_1> <a> <b> <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

$$p = \mathbf{a}\, t^3 + \mathbf{b}\, t^2 + \mathbf{c}\, t + \mathbf{d}$$

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_0> - 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 <X_0> <Y_0> <Z_0> <A_0> <B_0> <C_0> <U_0> <V_0> <X_1> <Y_1> <Z_1> <A_1> <B_1> <C_1> <U_1> <V_1> <a> <b> <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.

($X_0,Y_0,Z_0,A_0,B_0,C_0,U_0,V_0$) - beginning of path

($X_1,Y_1,Z_1,A_1,B_1,C_1,U_1,V_1$) - end of path

3rd order parametric equation where

$$p = \mathbf{a}\, t^3 + \mathbf{b}\, t^2 + \mathbf{c}\, t + \mathbf{d}$$

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

<U_0> - U begin point

<V_0> - V begin point

<X_0> - 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

<U_1> - U end point

<V_1> - 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

LinearP *<X_0> <Y_0> <Z_0> <A_0> <B_0> <C_0> <U_0> <V_0> <XP_0> <YP_0> <ZP_0> <AP_0> <BP_0> <CP_0> <UP_0> <VP_0> <X_1> <Y_1> <Z_1> <A_1> <B_1> <C_1> <U_1> <V_1> <XP_1> <YP_1> <ZP_1> <AP_1> <BP_1> <CP_1> <UP_1> <VP_1> <a> <b> <c> <d> <t_F>*

Description

Place linear (in 16 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, v* and prime axes *xp, yp, zp, ap, bp, cp, up, vp* 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 LinearEx command may be sent directly, the LinearEx command is normally generated automatically to perform a planned trajectory by the coordinated motion library or GCode, however currently the GCode Interpreter only supports 8 axes of simultaneous motion.

($X_0,Y_0,Z_0,A_0,B_0,C_0,U_0,V_0,XP_0,YP_0,ZP_0,AP_0,BP_0,CP_0,UP_0,VP_0$) - beginning of path

($X_1,Y_1,Z_1,A_1,B_1,C_1,U_1,V_1,XP_1,YP_1,ZP_1,AP_1,BP_1,CP_1,UP_1,V_{P1}$) - end of path

3rd order parametric equation where
$p = \mathbf{a}\, t^3 + \mathbf{b}\, t^2 + \mathbf{c}\, t + \mathbf{d}$

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

<U_0> - U begin point

<V_0> - V begin point

<XP_0> - XP begin point

<YP_0> - YP begin point

<ZP_0> - ZP begin point

<AP_0> - AP begin point

<BP_0> - BP begin point

<CP_0> - CP begin point

<UP_0> - UP begin point

<VP_0> - VP 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

<U_1> - U end point

<V_1> - V end point

<XP_1> - XP end point

<YP_1> - YP end point

<ZP_1> - ZP end point

<AP_1> - AP end point

<BP_1> - BP end point

<CP_1> - CP end point

<UP_1> - UP end point

<VP_1> - VP end point

<*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 - Move all 16 axes from 0 to 1 over 1 second

LinearP 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 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 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 <X_0> <Y_0> <Z_0> <A_0> <B_0> <C_0> <X_1> <Y_1> <Z_1> <A_1> <B_1> <C_1> <*a*> <*b*> <*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 - Move all 6 axes from 0 to 1 over 1 second

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

LinearHexEx *<X_0> <Y_0> <Z_0> <A_0> <B_0> <C_0> <U_0> <V_0> <X_1> <Y_1> <Z_1> <A_1> <B_1> <C_1> <U_1> <V_1> <a> <b> <c> <d> <t_F>*

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 – Move all 8 axes from 0 to 1 over 1 second

LinearHexEx 0 0 0 0 0 0 0 0 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 0 0 3F800000 0 3F800000

LinearHexP *<X0> <Y0> <Z0> <A0> <B0> <C0> <U0> <V0> <XP0> <YP0> <ZP0> <AP0> <BP0> <CP0> <UP0> <VP0> <X1> <Y1> <Z1> <A1> <B1> <C1> <U1> <V1> <XP1> <YP1> <ZP1> <AP1> <BP1> <CP1> <UP1> <VP1> <a> <b> <c> <d> <tF>*

Description

Place linear (in 16 dimensions) interpolated move into the coordinated motion buffer. This command is exactly the same as the LinearP command above, except all 37 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 - Move all 16 axes from 0 to 1 over 1 second

LinearHexP 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 0 0 3F800000 0 3F800000

LHex1 <X_1> <Y_1> <Z_1> <A_1> <B_1> <C_1> <a> <b> <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 LinearHex command above, except the beginning point is not specified and is assumed to be the endpoint of the previous LinearHex or LHex1 command. See also also KMotion Coordinated Motion.

Parameters

See above.

Example

LinHex1 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 0 0 3F800000 0 3F800000

3F800000

LhexEx1 <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 LinearHexEx command above, except the beginning point is not specified and is assumed to be the endpoint of the previous LinearHexEx or LHexEx1 command. See also also KMotion Coordinated Motion.

Parameters

See above.

Example - Move from previous point to 1 on all 8 axes over 1 second

LinHexEx1 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 0 0 3F800000 0 3F800000

LHexP1 *<X1> <Y1> <Z1> <A1> <B1> <C1> <U1> <V1> <XP1> <YP1> <ZP1> <AP1> <BP1> <CP1> <UP1> <VP1> <a> <b> <c> <d> <tF>*

Description

Place linear (in 16 dimensions) interpolated move into the coordinated motion buffer. This command is exactly the same as the LinearHexP command above, except the beginning point is not specified and is assumed to be the endpoint of the previous LinearHexP or LHexP1 command. See also KMotion Coordinated Motion.

Parameters

See above.

Example - Move from previous point to 1 on all 16 axes over 1 second

LinHexP1 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 3F800000 0 0 3F800000 0 3F800000

LHex2 *<a> <b> <c> <d> <t_F>*

Description

Place linear (in 6 or 8 dimensions) interpolated move into the coordinated motion buffer. This command is exactly the same as the LinearHex or LinearHexEx command above, except neither the beginning or ending point is specified and is assumed to be the same as the most recent LinearHex, LinearHexEx, LHex1, or LHex2,   command. This command can be used when there are more than one phases (ie Jerk, acceleration, constant velocity, etc. that occur along a single linear segment). See also KMotion Coordinated Motion.

Parameters

See above.

Example - move from beginning to end over 1 second

LinHex2 0 0 3F800000 0 3F800000

LHexEx1 *<X_1> <Y_1> <Z_1> <A_1> <B_1> <C_1> <U_1> <V_1> <a> <b> <c> <d> <t_F>*

Description

Place linear (in 8 dimensions) interpolated move into the coordinated motion buffer. This command is exactly the same as the LinearHexEx command above, except the beginning point is not specified and is assumed to be the endpoint of the previous LinearHexEx or LHexEx1 command. See also KMotion Coordinated Motion.

Parameters

See above.

Example

LinHexEx1 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

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

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

Example

LoadData 80030000 4

FF FF FF FF

LoadFlash<*H*> <*N*>

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

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

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

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 my 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

Move*AtVel*<*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

MoveAtVelAccel<*N*>=<*D*> <V><A>

Description

Move axis N to absolute coordinates D at the specified Velocity and Acceleration.

Parameters

<N>
Selected Axis for command. Valid range 0...7.

<D>

Coordinates in position units. Valid range - any.

<V>

Desired Velocity for the Motion. Valid range - any.

<A>

Desired Acelleration for the Motion. Valid range - any.

Example

MoveAtVelAccel0=100.1 30.0 2000.0

MoveExp<*N*>=<*D*> <*T*>

Description

Moves axis in an exponential manner toward the Destination using Time Constant T. The velocity of motion will be proportional to the distance from the Destination. The distance to the Destination will be reduced by 63% (1/e) every Time Constant, T. The Axis should be already enabled. Honors the Vel and Accel axis parameters.

Parameters

<N>
Selected Axis for command. Valid range 0...7.

<D>

Destination in position units. Valid range - any.

<T>

Time Constant Tau in seconde. Valid range - any positive number.

Example

MoveExp0=1000 0.1

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

Move*RelAtVel*<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

MoveRelAtVelAccel*<N>*=*<D> <V><A>*

Description

Move axis N relative to current destination D at the specified Velocity and Acceleration.

Parameters

<N>
Selected Axis for command. Valid range 0...7.

<D>

Coordinates in position units. Valid range - any.

<V>

Desired Velocity for the Motion. Valid range - any.

<A>

Desired Acceleration for the Motion. Valid range - any.

Example

MoveAtVelAccel0=100.1 30.0 2000.0

MoveRelAtVelAccelSoft*<N>*=*<D> <V><A>*

Description

Move axis N relative to current destination D at the specified Velocity and Acceleration. Limit to Soft Limit range.

Parameters

<N>
Selected Axis for command. Valid range 0...7.

<D>

New *destination* in position units. Valid range - any.

<V>

Desired Velocity for the Motion. Valid range - any.

<A>

Desired Acceleration for the Motion. Valid range - any.

Example

MoveRelAtVelAccelSoft0=100.1 30.0 2000.0

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

Description

Move the 6 axes defined to be X, Y, Z, A, B, C (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.

<B>

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 (from PC_DSP.h):

```
#define NO_OUTPUT_MODE 0
#define MICROSTEP_MODE 1
#define DC_SERVO_MODE 2
#define BRUSHLESS_3PH_MODE 3
#define BRUSHLESS_4PH_MODE 4
#define DAC_SERVO_MODE 5
#define STEP_DIR_MODE 6
#define CL_STEP_DIR_MODE 7
#define CL_MICROSTEP_MODE 8
```

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 or Virtual IO bit 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. Accepted range - 0...2047

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 or Virtual I/O bit to high (1).

Parameters

<N>

Bit number to set. Accepted range 0...2047

Example

SetBit0

SetBitBuf<*N*>

Description

Inserts into the coordinated move buffer a command to set an I/O bit (actual I/O 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. Depending on the type of I/O, it may not be possible to change direction as some I/O bits are strictly inputs, and some are strictly outputs.

See also Digital I/O Screen.

Parameters

<N>
Bit number to assign. Accepted range 0...2047

<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>*

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, SetGatherHex

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

SetFRO *<F>*

Description

Sets Hardware FRO (Feed Rate Override) in KFLOP which is the rate that the Coordinated Motion Buffer is executed. A value of 1.0 = Normal Feed Rate = real time = an advance of 90us of time every 90us Servo Sample Period.

A negative FRO value will cause the Coordinated Motion Buffer to execute in reverse up until the beginning or until the point where Coordinated Motion Buffer data has been lost due to buffer wrapping (MAX_SEGMENTS is currently ~35,000 segments). When approaching the point where previous data was lost, the FRO will be automatically reduced to zero in order to avoid an abrupt stop. This will not occur (and should not be necessary) when approaching the actual beginning of the buffer because normal acceleration from a stop should exist. In this case Time will stop abruptly when the beginning of the buffer is reached.

In order to avoid an instantaneous change in velocity the FRO will be ramped from the current rate to the specified rate. This command uses a default ramp rate that has been determined based on the Max Allowed Velocities, Accelerations, and Jerks of all the currently defined Coordinate Motion System Axes Channels. In order to specify a different rate the SetFROwRate command may be used.

This command will not alter the rate of execution if the FeedHold mechanism is currently in effect. See StopImmediate. However the specified speed will be saved so that if FeedHold is eventually released, the rate will resume to this specified speed. To change the FRO while in FeedHold use the SetFROTemp or SetFROwRateTemp commands instead. Those commands were intended to be used while in Feed Hold and will not alter the rate that will be resumed after Feed Hold is released.

Parameters

<F>

Desired FRO Value. 1.0 corresponds to normal Real Time, 0.0 corresponds to fully stopped, negative values drive time in reverse. Valid range -100...+100

Example

SetFRO 1.2

SetFROTemp <*F*>

Description

Sets Hardware FRO (Feed Rate Override) in KFLOP which is the rate that the Coordinated Motion Buffer is executed. A value of 1.0 = Normal Feed Rate = real time = an advance of 90us of time every 90us Servo Sample Period.

This command is intended for temporary FRO changes while in Feed Hold.

See SetFRO for additional Information.

Parameters

<F>

Desired FRO Value. 1.0 corresponds to normal Real Time, 0.0 corresponds to fully stopped, negative values drive time in reverse. Valid range -100...+100

Example

SetFROTemp -0.2

SetFROwRate <*F*> <R>

Description

Sets Hardware FRO (Feed Rate Override) in KFLOP which is the rate that the Coordinated Motion Buffer is executed. A value of 1.0 = Normal Feed Rate = real time = an advance of 90us of time every 90us Servo Sample Period. This command functions the same as the command SetFRO with the exception that the rate at which the FRO will be ramped to the new FRO may be controlled. The ramp rate (rate-of-change-of-rate-of-time) to be used is determined from a user supplied Time Parameter. The Time to ramp from FRO=0. to FRO=1.0. See SetFRO for more information.

Parameters

<F>

Desired FRO Value. 1.0 corresponds to normal Real Time, 0.0 corresponds to fully stopped, negative values drive time in reverse. Valid range -100...+100

<R>

Time to ramp from FRO=0.0 to FRO=1.0 Valid range any positive number.

Example

SetFROwRate 1.2 0.5

SetFROwRateTemp *<F>* <R>

Description

Sets Hardware FRO (Feed Rate Override) in KFLOP which is the rate that the Coordinated Motion Buffer is executed. A value of 1.0 = Normal Feed Rate = real time = an advance of 90us of time every 90us Servo Sample Period. This command functions the same as the command SetFROTemp with the exception that the rate at which the FRO will be ramped to the new FRO may be controlled. The ramp rate (rate-of-change-of-rate-of-time) to be used is determined from a user supplied Time Parameter. The Time to ramp from FRO=0. to FRO=1.0.

This command is intended for temporary FRO changes while in Feed Hold.

See SetFRO for additional Information.

Parameters

<F>

Desired FRO Value. 1.0 corresponds to normal Real Time, 0.0 corresponds to fully stopped, negative values drive time in reverse. Valid range -100...+100

<R>

Time to ramp from FRO=0.0 to FRO=1.0 Valid range any positive number.

Example

SetFROwRateTemp -0.2 0.5

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

SetIpAddr<*D*><*D*><*D*><*D*> (Kogna only)

Description

Set board's Ethernet IP Address.

Parameters

<D>

Each D is 0 ... 255.

Example

SetSerialNumber 192.168.10.9

SetKognaPWMEne <*N*>=<*N*> (Kogna only)

Description

Set Kogna PWM. Enable channels 0-7. 1=Enable 0=Disable.

Parameters

<N>

PWM channel number. Valid range 0...7.

Example

SetKognaPWMEn0=1

SetKognaPWMLength<*N*>=<*N*> (Kogna only)

Description

Set Kogna PWM Pulse Length channels 0-7. 0-255 counts.

Parameters

<N>

PWM channel number. Valid range 0...7.

Example

SetKognaPWMLength0=128

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 *flashed*, 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 *flashed*, 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

SetRapidFRO <*F*>

Description

Sets Hardware RFRO (Rapid Feed Rate Override) in KFLOP which is the rate that the Coordinated Motion Buffer is executed. A value of 1.0 = Normal Feed Rate = real time = an advance of 90us of time every 90us Servo Sample Period.

Note: KFLOP Maintain separate Rate Overides for Rapid motion vs normal Feed Motion. Commands (BegRapidBuf and EndRapidBuf) inserted into the Coordinated Motion Buffer determine what type of motion is currently in progress and which Override is to be used.

A negative RFRO value will cause the Coordinated Motion Buffer to execute in reverse up until the beginning or until the point where Coordinated Motion Buffer data has been lost due to buffer wrapping (MAX_SEGMENTS is currently ~35,000 segments). When approaching the point where previous data was lost, the RFRO will be automatically reduced to zero in order to avoid an abrupt stop. This will not occur (and should not be necessary) when approaching the actual beginning of the buffer because normal acceleration from a stop should exist. In this case Time will stop abruptly when the beginning of the buffer is reached.

In order to avoid an instantaneous change in velocity the RFRO will be ramped from the current rate to the specified rate. This command uses a default ramp rate that has been determined based on the Max Allowed Velocities, Accelerations, and Jerks of all the currently defined Coordinate Motion System Axes Channels. In order to specify a different rate the SetRapidFROwRate command may be used.

This command will not alter the rate of execution if the FeedHold mechanism is currently in effect. See StopImmediate. However the specified speed will be saved so that if FeedHold is eventually released, the rate will resume to this specified speed. To change the FRO while in FeedHold use the SetFROTemp or SetFROwRateTemp commands instead. Those commands were intended to be used while in Feed Hold and will not alter the rate that will be resumed after Feed Hold is released.

Parameters

<F>

Desired RFRO Value.   1.0 corresponds to normal Real Time, 0.0 corresponds to fully stopped, negative values drive time in reverse. Valid range -100...+100

Example

SetRapidFRO 1.2

SetRapidFROwRate *<F>* *<R>*

Description

Sets Hardware RFRO (Rapid Feed Rate Override) in KFLOP which is the rate that the Coordinated Motion Buffer is executed. A value of 1.0 = Normal Feed Rate = real time = an advance of 90us of time every 90us Servo Sample Period. This command functions the same as the command SetRapidFRO with the exception that the rate at which the RFRO will be ramped to the new RFRO may be controlled. The ramp rate (rate-of-change-of-rate-of-time) to be used is determined from a user supplied Time Parameter. The Time to ramp from FRO=0. to FRO=1.0. See SetRapidFRO for more information.

Note: KFLOP Maintain separate Rate Overrides for Rapid motion vs normal Feed Motion. Commands (BegRapidBuf and EndRapidBuf) inserted into the Coordinated Motion Buffer determine what type of motion is currently in progress and which Override is to be used.

Parameters

<F>

Desired RFRO Value. 1.0 corresponds to normal Real Time, 0.0 corresponds to fully stopped, negative values drive time in reverse. Valid range -100...+100

<R>

Time to ramp from RFRO=0.0 to RFRO=1.0 Valid range any positive number.

Example

SetRapidFROwRate 1.2 0.5
1.2 0.5

SetSerialNumber<*N*> (Kogna only)

Description

Set board's Serial Number.

Parameters

<N>

Decimal number Valid Range 0...4095.

Example

SetSerialNumber 123

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 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. Accepted range 0...2047

<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 (actual IO bits must be defined as an output, see SetBitDirection).

Parameters

<N>

Bit number to set. Accepted range 0...2047

<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

SoftLimitNeg<*N*>=<*M*>

or

SoftLimitPos<*N*>

Description

Command to set or display the Negative Software Limit of Travel. Soft Limits will prevent motion in the same manner as a Hardware Limit with the Stop Movement Action Selected. This occurs regardless of the Action Type Selected for the Hardware Limit Switches. To disable Soft Limits set them to a huge range which could never occur. Soft Limits prevent motion within KFLOP when Jogging, moving and so forth. They also are are uploaded by Applications such as KMotionCNC and used to prevent motion during Trajectory Planning. The Negative Soft Limit is used to prevent motion beyond a limit in the negative direction. The Negative Soft Limit does not necessarily need to be negative. See also SoftLimitPos.

Parameters

<N>

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

<M>

Maximum Negative Limit. Valid range - any value. Set to a large value to disable.

Example

SoftLimitNeg0=-1000000.0
or
SoftLimitNeg0

SoftLimitPos<*N*>=<*M*>

or

SoftLimitPos<*N*>

Description

Command to set or display the Positive Software Limit of Travel. Soft Limits will prevent motion in the same manner as a Hardware Limit with the Stop Movement Action Selected. This occurs regardless of the Action Type Selected for the Hardware Limit Switches. To disable Soft Limits set them to a huge range which could never occur. Soft Limits prevent motion within KFLOP when Jogging, moving and so forth. They also are are uploaded by Applications such as KMotionCNC and used to prevent motion during Trajectory Planning. The Positive Soft Limit is used to prevent motion beyond a limit in the positive direction. The Positive Soft Limit does not necessarily need to be positive. See also SoftLimitNeg.

Parameters

<N>

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

<M>
Maximum Positive Limit. Valid range - any value. Set to a large value to disable.

Example

SoftLimitPos0=1000000.0
or
SoftLimitPos0

SPISetMode<*N*>=<*M*> (Kogna only)

Description

Set SPI Pin Mode mux channels 0-5. 1=GPIO 0=HRPWM 2=I2C.

Parameters

<N>

Specifies the Pin.

<M>

Specifies the Mode.

Example

SPISetMode0=1

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

StopImmediate<*M*>

Description

Controls the Feedhold Mechanism for the set of coordinated motion Axes. This command can be used to feedhold (bring to an immediate stop) the set of axes, Resume from a feedhold, or clear the feedhold state. This command can stop the set of axes regardless of whether the current motion in progress is due to coordinated motion (Interpolated Linear or Arc) or independent axes motions (Rapids). The current state can be obtained using the GetStopState command.

Parameters

<M>

Mode

0 - Stops the axes motion (equivalent to User C Program function StopCoordinatedMotion)
1 - Resumes the axes motion (equivalent to User C Program function ResumeCoordinatedMotion)
2 - Clears the Feed hold state (equivalent to User C Program function ClearStopImmediately)

Example

StopImmediate0

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

USB (Kogna only)

Description

Commands from USB Mode.

Parameters

None.

Example

USB

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

WaitBitBuf<*N*>

Description

Inserts into the coordinated move buffer a command to wait for an I/O bit to be at a high level. This command is useful for synchronizing motion to external events without any PC delays.

This command can be inserted into the Coordinated motion buffer from KMotionCNC GCode using the special comment command format of:

(BUF,WaitBitBuf46)

Parameters

<N>

Bit number to wait to be high. Accepted range 0...2047

Example

WaitBitBuf46

WaitNotBitBuf<*N*>

Description

Inserts into the coordinated move buffer a command to wait for an IO bit to be at a low level. This command is useful for synchronizing motion to external events without any PC delays.

This command can be inserted into the Coordinated motion buffer from KMotionCNC GCode using the special comment command format of:

(BUF,WaitNotBitBuf46)

Parameters

<N>

Bit number to wait to be low. Valid range 0...2047

Example

WaitNotBitBuf46

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.

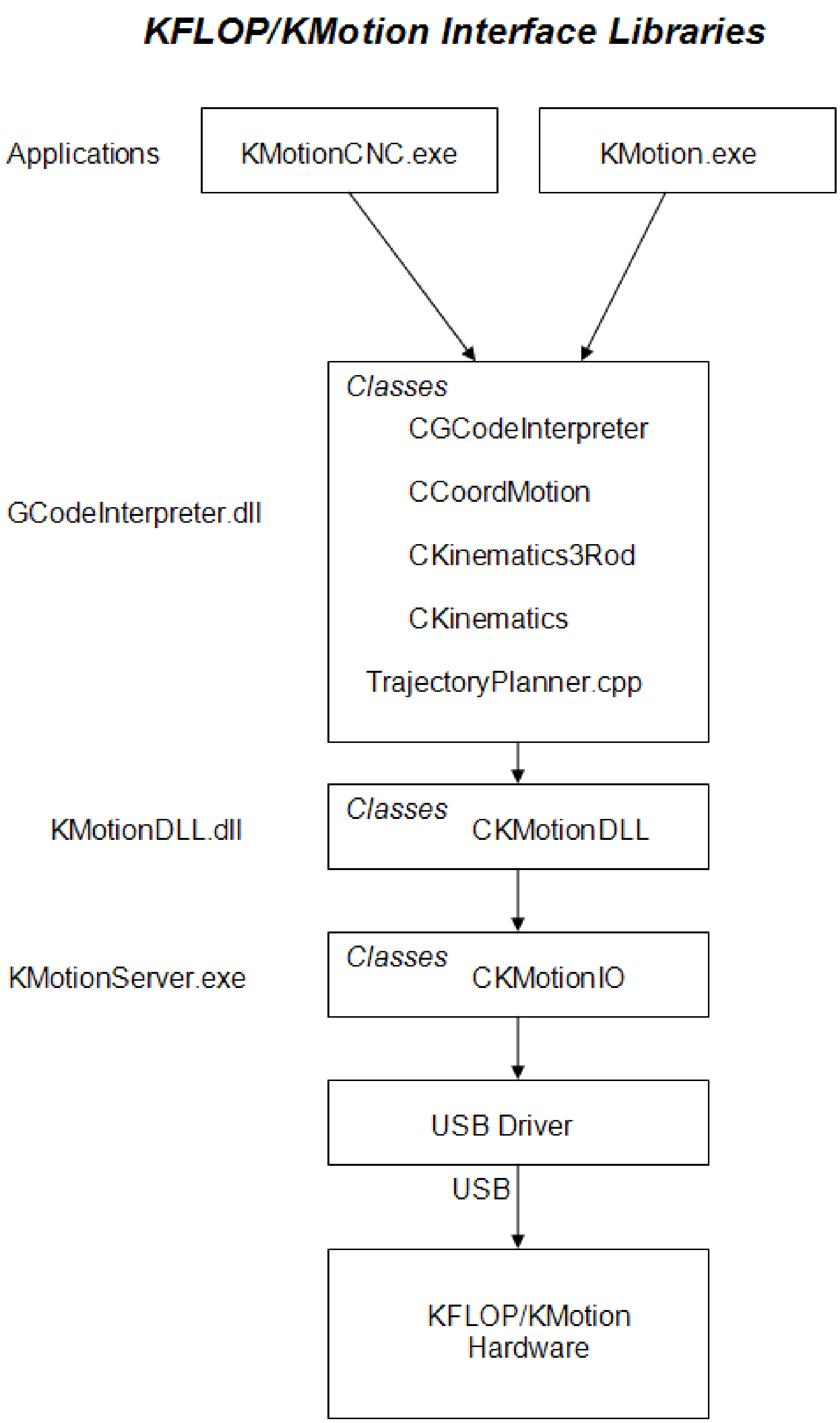
Parameters

<N>
Selected Axis for command. Valid range 0...7.

Example

Zero0

KFLOP/KMotion Interface Libraries



Using Multiple KFLOP/Kogna Boards

The ***KMotion*** Driver Library allows multiple PC processes (applications), each running multiple threads of execution, to communicate with multiple ***KFLOP/Kogna*** boards simultaneously. Each ***KFLOP/Kogna*** board is identified by a 32 bit integer.

A value of:

0 - Requests use of the first KFLOP or Kogna board that is discovered

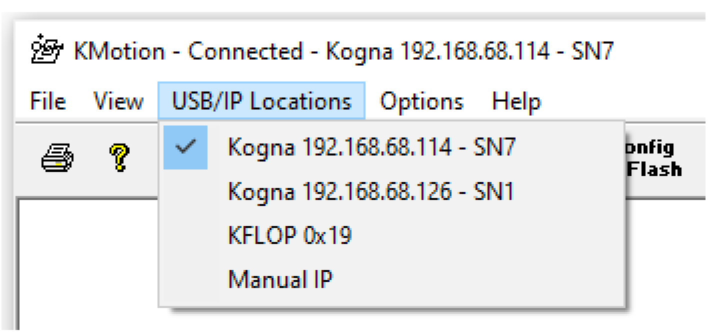
< 0x00FFFFFF - indicates a USB Location ID of a KFLOP

0xFF000XXX - specifies to use a Kogna board discovered that has Serial Number XXX

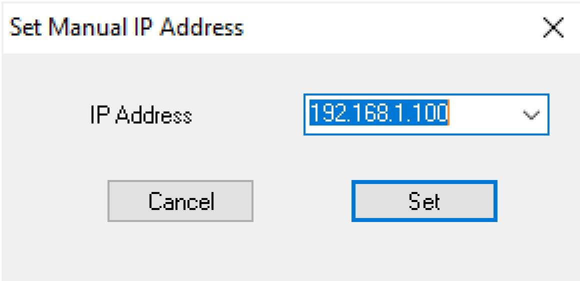
Otherwise - specifies a static IP Address (big endian) where 0xC0A80A64 would correspond to 192.168.10.100

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.

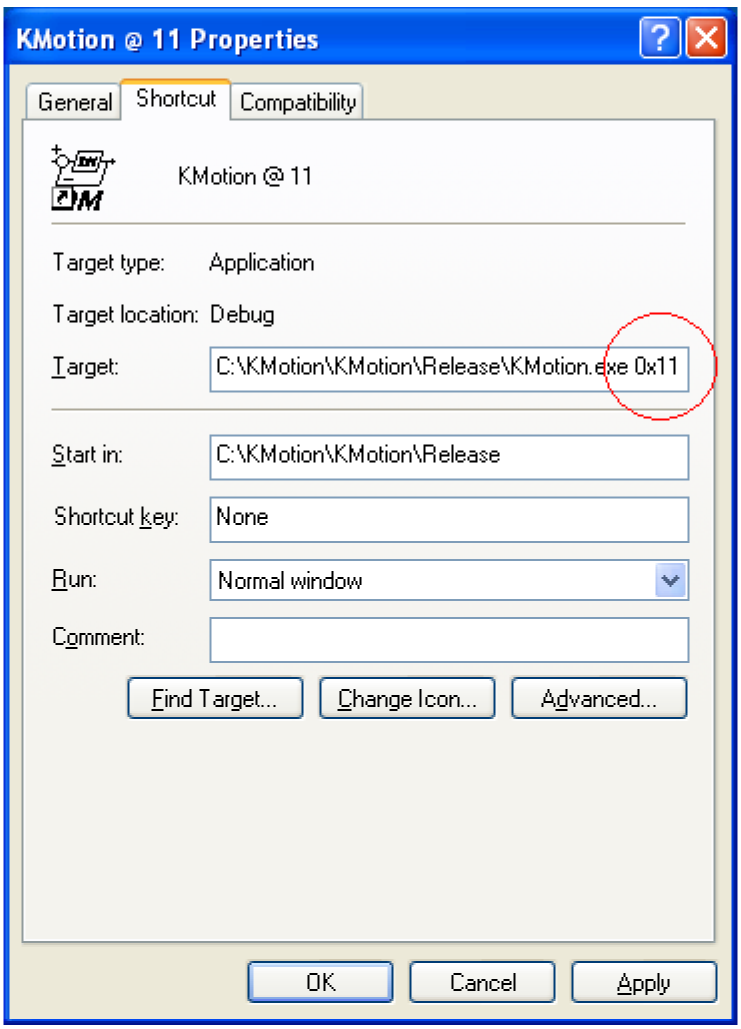
Selecting the USB/IP Locations menu of the ***KMotion*** Setup and Tuning application, will display a list of all currently connected ***KMotion*** boards (KFLOPs and/or Kognas). The Checkmark indicates which board the application is currently communicating with. To switch to a different board, select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



To enter a IP Address manually select Manual IP and the entry form below will be shown.



When launching the ***KMotion*** Setup and Tuning application, a command line parameter may be specified to connect to a specific USB/IP 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. A hexadecimal value preceded with 0x may be specified or an IP Address can be specified with the -i option such as -i 192.168.10.100



The ***KMotion*** Driver Library has a function to list the locations of all currently discovered ***KFLOP/Kogna*** boards. See:

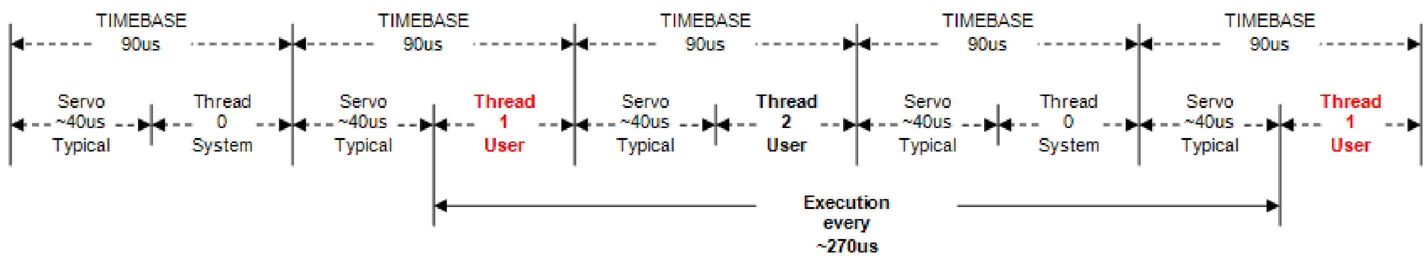
```
int ListLocations(int *nlocations, int *list);
```

The reurned list will contain all the KFLOP and Kogna's discovered. Each KFLOP discovered will have one entry containing the USB Location ID where it is connected. Each Kogna discovered will have 2 entries. The first as the Ethernet IP Address. The second will contain the Kogna's Serial Number in the format 0xFF000XXX where XXX is the hexadecimal serial number.

PC Applications specify the Board ID as a parameter when creating a CKMotionDLL object to communicate with a board. For example:

```
CKMotionDLL *KM = new CKMotionDLL(ID);
```

KFLOP/Kogna Preemptive Multitasking



KFLOP/Kogna uses a simple method of preemptive multitasking (multiple programs or "Threads" running at the same time). Each Thread consists of an area of memory where a program can be loaded, A CPU Stack for that program, and a potential time slice of the CPU. User Programs (Threads) and the System Thread context switch every Servo Interrupt and sequence in a round robin order.

The main thing to understand is that two programs that ever need to be running at the same time need to be assigned to different threads. The KFLOP/Kogna system Thread runs all the time in Thread #0. So Thread #0 may never be used. In a typical KMotionCNC system the Init.c program is usually assigned to thread #1 and runs forever. If other UserButtons (or MCodes) run C Programs that do something and terminate (Exec/Wait option) then these might all be assigned to use Thread#2. If your programs run for a longer time where a 3rd or 4th program might be Launched so they all overlap in time, then they must all be assigned to different Threads.

Most Applications have a Red Stop Button that stops all or most Threads from executing. For example KMotion.exe stops all User Threads. KMotionCNC stops all User Threads except Thread #1 so any critical monitoring program might be placed in Thread #1 in order to continue execution after an Emergency Stop.

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:

Period = (# User Threads + 1) * TIMEBASE

KFLOP/Kogna - RS232

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

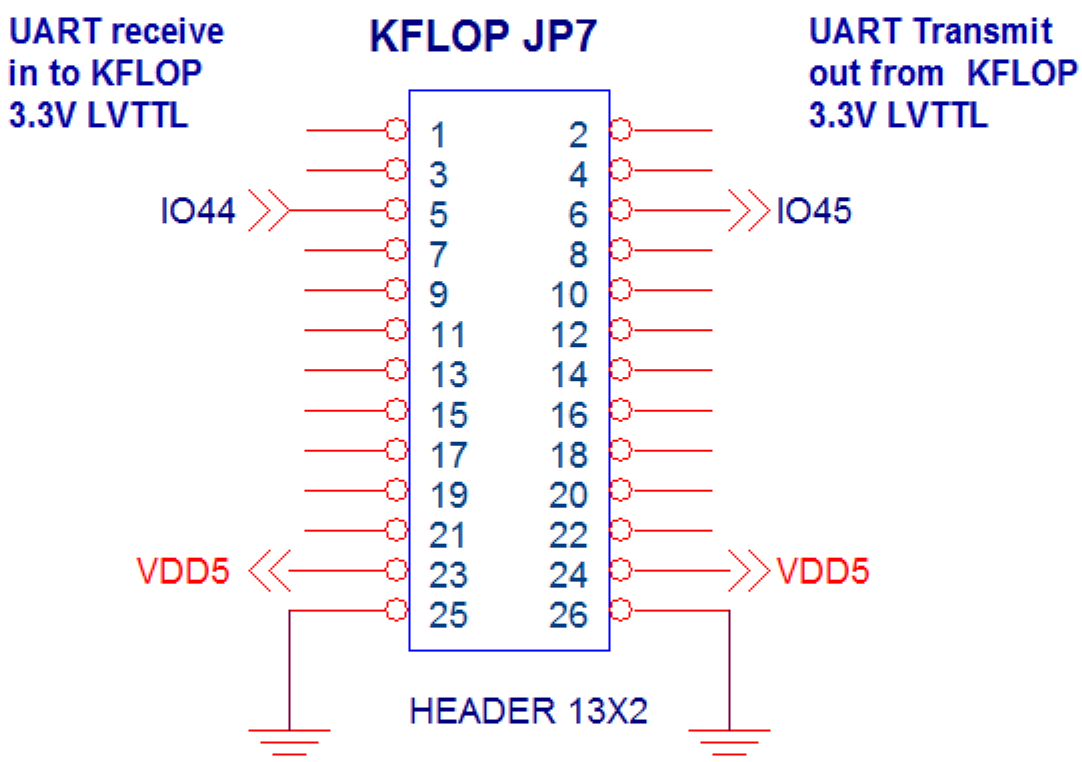
KFLOP/Kogna itself contains the serial communication UART but does not have the circuitry to drive or receive the signals at the +3 to 25V to -3 to -25V voltages levels specified by the RS232 standard. The transmit and receive signals to/from KFLOP/Kogna are 3.3V LVTTL logic signals. Where a low logic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

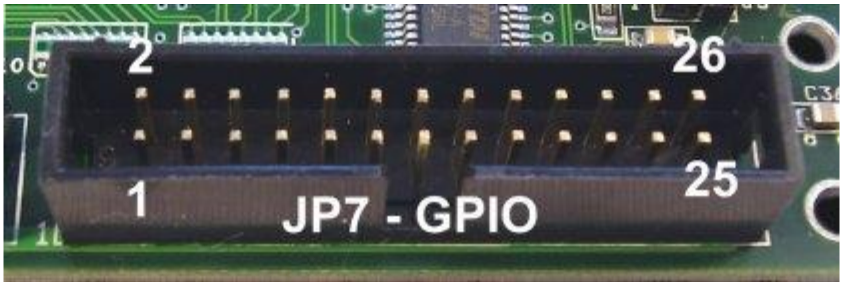
Note that the signals that the signals coming directly from KFLOP are not RS232 compatible. Connecting KFLOP/Kogna inputs directly to RS232 is likely to cause damage.

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 LVTTL to converter on board (see next section below).

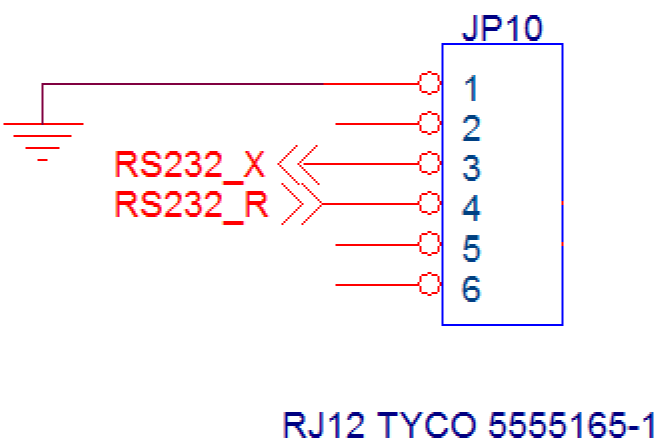




KFLOP/Kogna + Kanalog - RS232

Kanalog contains circuitry to convert the KFLOP/Kogna 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.



KFLOP/Kogna UART Software

The KFLOP/Kogna FPGA implements a programmable baud rate UART with double buffering of 1 character on transmit and receive. Currently a KFLOP/Kogna 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/Kogna User Programs with 1 active thread execute every 180us. (See here for more info). This allows rates up t0 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/Kogna 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)=2;
```

Note the techniques shown below may be used but a new simpler and buffered method is now available. See the \C Programs\RS232\BufferedRS232.c example

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)

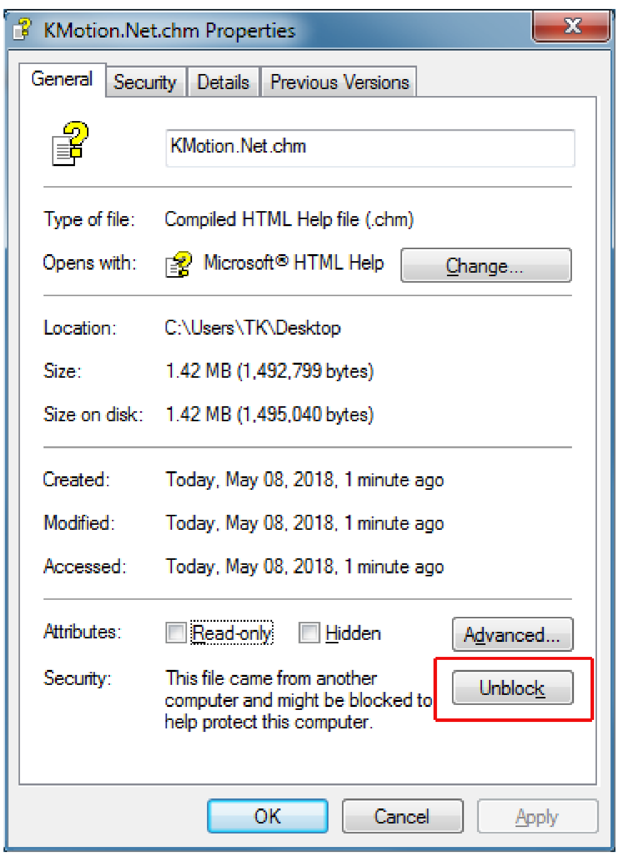
```
#include "KMotionDef.h"

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());
}
}
```

.NET Interface

Dynomotion's Motion Libraries provide a .NET interface. The .NET interface allows all the functionality of the KMotion Libraries to be imported and made accessible to your application with a simple reference to KMotion_dotNet.dll. .NET is a standard that works with many different Languages including Visual C#, Visual Basic, Visual C++, IronPython, LabView, and many others. Once the reference is included in the project all the classes, member functions, variable, and data types are accessible. Most languages will provide intellisense and auto complete for all the imported objects once a reference is established.

The functionality exposed by the KMotion .NET interfaces is documented in a Help file. This documentation is available in two formats. As an installed Windows Help file (KMotion_dotNet.dll) or as on-line pages view-able with a browser. The Installed Windows Help file is installed with KMotion at <installed>\KMotion_dotNet\Docs\Help\KMotion.Net.chm

You can also download the .NET Interface Windows Help from our website. You will need to save the file, then right-mouse-click it, and select “Properties” and “Unblock”:



KMotionCNC

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)

G17 Arcs in XY plane
G18 Arcs in XZ plane
G19 Arcs in YZ plane

G20 Inch units
G21 mm units

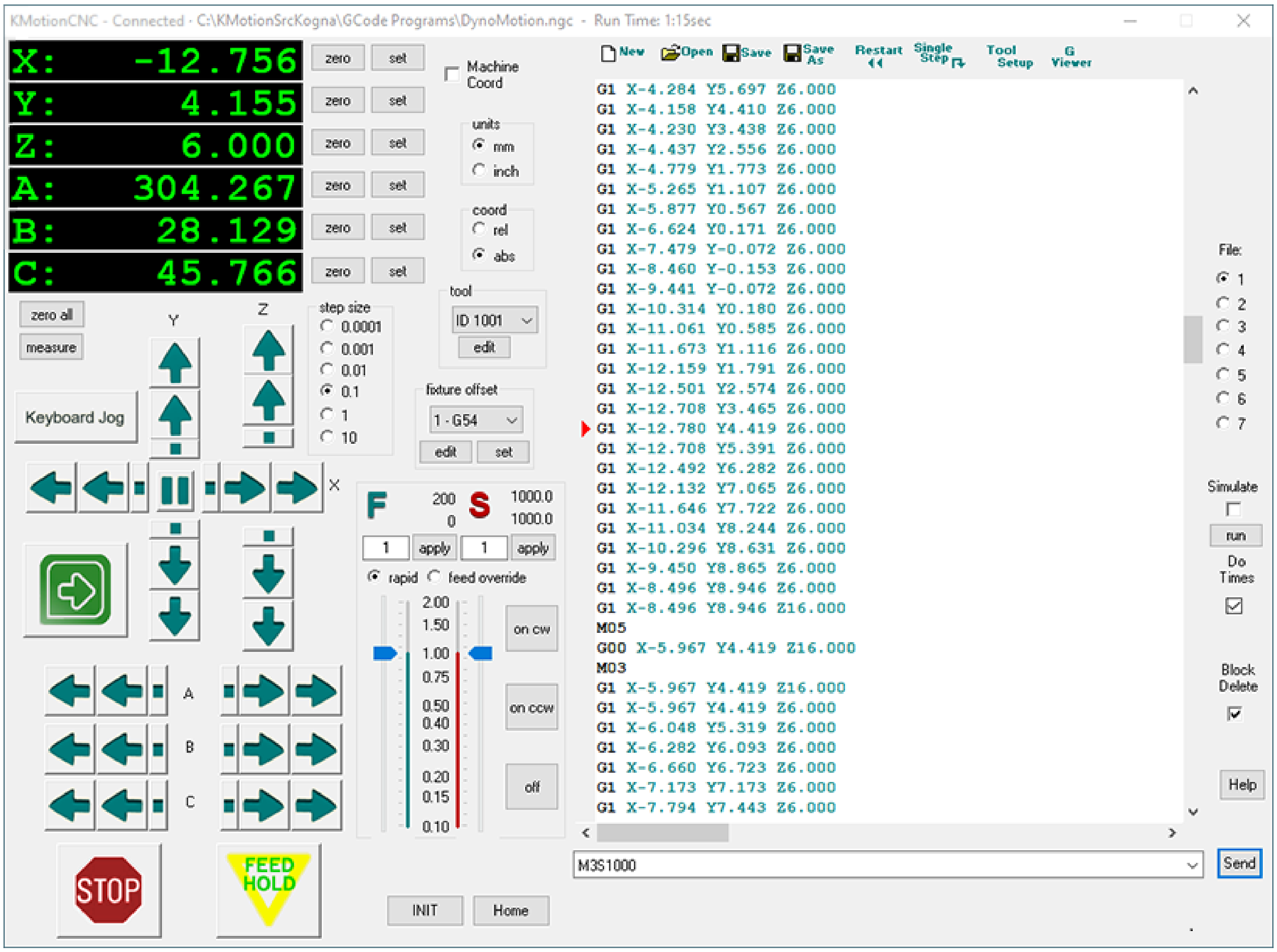
G28 Move to Reference Position #1
G30 Move to Reference Position #2

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)
G43.4 Hn (Tool #n length comp ON with RTCP ON)

G49 (Tool length and RTCP comp OFF)
G53 Absolute Coord
G54 Fixture Offset 1
G55 Fixture Offset 2

Other KMotionCNC Screens
[G Code Viewer Screen](#)
[G Code Viewer Setup Screen](#)
[Tool Setup Screen](#)



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](#).

Right-mouse-clicking the title bar in KMotionCNC brings up a context menu for access to the Tool Setup Screens as well as the version (under "About KMotionCNC").
See the Quick Reference at left for commonly used G Code commands.

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

G84 Z-1. R.1 Q.1 F.0625 (Rigid Tap see wiki)

G90 Absolute Coordinates
G91 Relative Coordinates

G92/G52 Set Global Offset
G92 X0Y0Z0
G92.1 Clear Global Offset

G92.2 Clear Leave Vars
G92.3 Load Vars

G94 Feed per min
G95 Feed per Rev

G96 Dmax CSS mode
G97 Spindle RPM mode

M Codes:
M0 (Program Stop)
M1 (Opt Program Stop)
M2 (Program End+Reset)
M3 Spindle CW
M4 Spindle CCW
M5 Spindle Stop
M6 Tool Change
M7 Mist On
M8 Flood On
M9 Mist/Flood Off
M30 (Program End)
M47 Loop to Beginning
M98 Pxxx Call Subroutine
M99 Return from Sub

Other Codes:
F (Set Feed rate in/min or mm/min)

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.

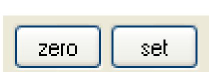
Orange - indicates normal status, hardware is connected, axis is enabled, and the displayed position is the current tool position in Raw Machine Coordinates (G53 without any Global Offset (G92) or Fixture Offset (G54+).

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

Cyan - indicates normal status, hardware is connected, axis is enabled, and the displayed position is the current measured tool position in GCode coordinates. Available when Encoders or other feedback device is used and the Encoder mode is selected on the Tool Setup | Trajectory Planner Screen.

Zero/Set Origin



Allows zeroing or setting the respective axis. This is accomplished by adjusting the Global Offsets (G52/G92) or by adjusting the currently selected Fixture Offset. Which offset is adjusted is determined by the Tool Setup Parameter described here. The original Raw Machine Coordinates will remain unchanged, so the effect of these

S (Spindle Speed)
D (Tool)
O Subroutine Label

Comments:

(Simple Comment)
(MSG,OK toContinue?)
(CMD,EnableAxis0)
(BUF,SetBitBuf29)

More GCodes

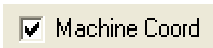
Keyboard Hot Keys

ESC - Stop
F2 - KeyJog on/off
F3 - FeedHold
F5 - Run/Halt
F9 - Spindle On CW
F10 - Spindle On CCW
F11 - Spindle Off

KeyJog Mode
(shift=fast)
(ctrl=step)
x y - Arrow Keys
z - page up/down
A - num - +

operations will only be apparent if displaying Machine Coordinates is de-selected, see below. For further information see system offsets.

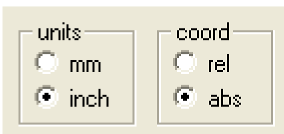
Coordinate System Origin



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



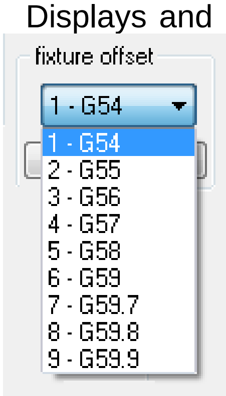
Displays the current mode of the G code interpreter.

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:



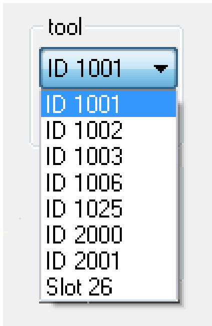
G10 L2 P3 X10.0 Y20.0 Z30.0 which sets Fixture Offset #3 to (10,20,30)

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).

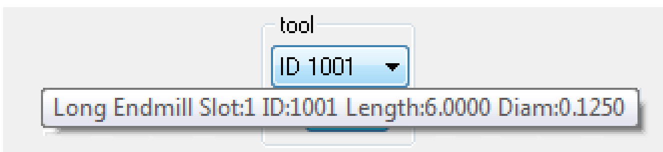
The “set” button can be used to automatically compute and change the current fixture offset so that the current position becomes the new origin.

The “edit” button can be used to view, change, or save the fixture offsets. See EditFixtureOffsets.

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. Tools can be selected by either 4 digitID or 2 digit Slot numbers. The Dropdown displays the available Tools defined in the tool table. If an ID is defined it will be displayed otherwise the slot number will be displayed.



Hovering the mouse over the tool selector will display a tool tip with information regarding the tool. Comment, Slot, ID, etc…

Besides changing tools with the T command, D and H commands are used to apply tool properties from the tool table.

Tool Numbers can be referenced by either Tool ID as a 4 digit (or higher) number or by Tool Slot as a 2 digit number.

A tool definition consists of a Tool Slot, Tool ID, the Length of the tool, the Diameter of the tool, XY offsets, a descriptive Comment, and a VRML Image File. All parameters are optional as long as either a Slot or ID is specified for each tool.

Note that the Tool Length can also be considered as a Z offset (Lathes often use this terminology).

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

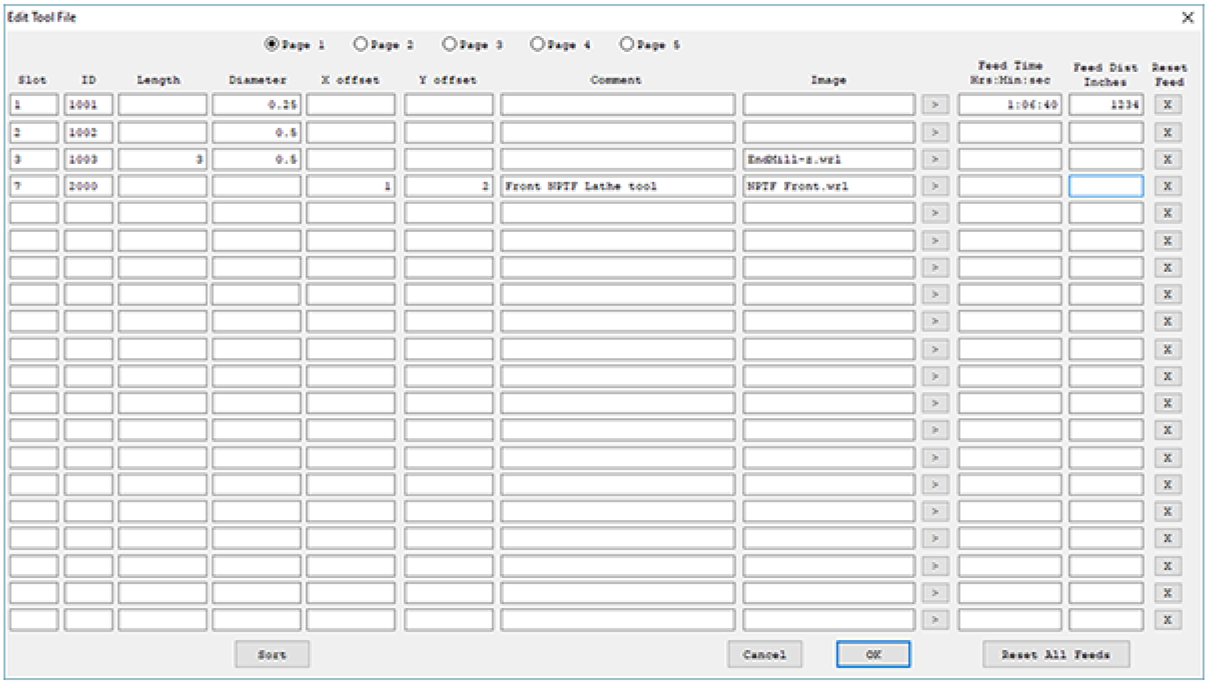
Executing the **H#** command will select which Tool parameters are to be used for XY compensation (**G43,G49**).

The Tool definitions are saved in a text file that is selected on the ToolSetup Screen. See below for an example Tool Table. There are 6 numeric and 2 alphanumeric strings. The strings, including empty strings, must be enclosed in quotes:

```
SLOT   ID   LENGTH     DIAMETER XOFFSET  YOFFSET  COMMENT                   IMAGE
1      1001 6.000000 0.125000 0.000000 0.000000 "first tool"              "EndMill-z 1in .125.wrl"
2      1002 2.000000 0.500000 0.000000 0.000000 ""                        "EndMill-z 2in .500.wrl"
0      1003 3.000000 0.500000 0.000000 0.000000 ""                        "EndMill-z.wrl"
7      1006 6.000000 0.000000 0.000000 0.000000 ""                        ""
25     1025 0.000000 0.000000 0.000000 0.000000 ""                        ""
```

```
0      2000 0.000000 0.000000 0.000000 0.000000 "Front NPTF Lathe tool"  "NPTF Front.wrl"
0      2001 0.000000 0.000000 0.000000 0.000000 "Back VNMG Lathe Tool"   "VNMG-1 back.wrl"
26     0    1.000000 0.250000 0.000000 0.000000 ""                       ""
```

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



The Tool Edit Screen displays 20 tools per page over 5 pages for a total of 99 possible tools.

The normal idea is that all information regarding a tool is linked to a Tool ID. For a particular GCode Job specific Tools can be loaded into specific Tool Changer Slots and the corresponding slot numbers can be edited to reflect which tools are currently loaded.

Note: When editing the tool table values there is an option to have current Tool Length/Offset applied immediately (see here) and the tool reloaded when the tool screen is closed.

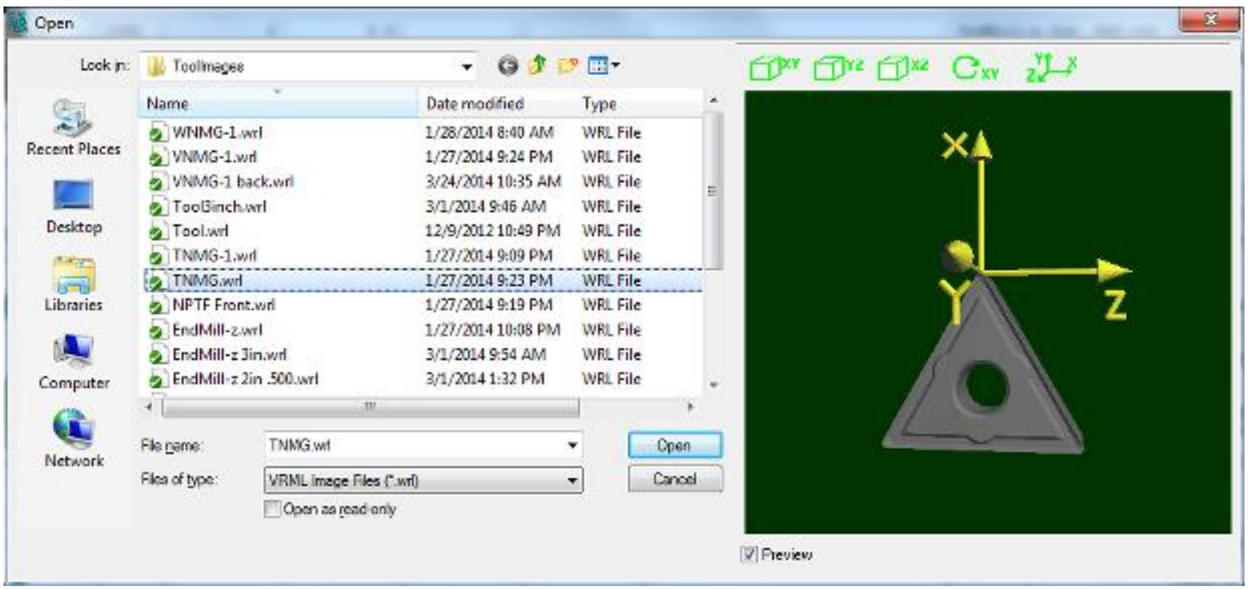
A “Sort” button is available that will sort the Tool Table order first by Tool ID and the by Slot number.

Feed Times / Feed Distances

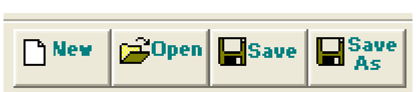
This new feature (introduced in KMotion 5.2.0) is used to keep track of how long a tool was actually used cutting (Feed). In GCode there are two types of moves. G0 rapids where the tool basically jumps from one place to another between cuts. And G1 G2 G3 feeds where the tool is moved at a constant feed rate usually cutting through material. Feed Times/Feed Distances allows the user to keep track of how long and how far a tool has been used. At some point an Operator might decide it needs to be replaced. At that time the values will be reset.

The Tool Image file refers to a VRML 3D model of the Tool that will be used to display the tool when shown in the GViewer Screen. All image files are normally located into the <Install>\KMotion\Data\ToolImages Folder. VRML (Virtual Reality Modeling Language) files normally have a .wrl file type extension. If located in the standard location no path is required to be specified. Pushing the Browse Button 

will bring up a File Selector Dialog with Preview that will display the available Tool Images as well as the Tool Image origin. Note that Mouse Click Right, Left, Both can be used to manipulate the View in the same manner as in the GViewer Screen. For creating your own Tool Image files see information here.



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

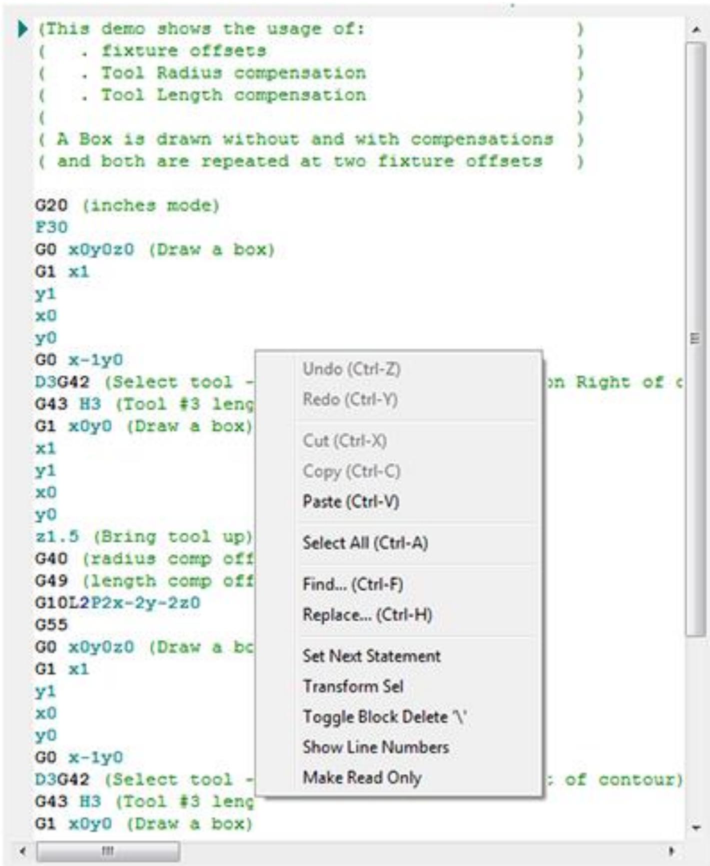
This group of pushbuttons allow the control of G Code execution. ***Restart*** will reset the instruction pointer  to the first line of the file. ***Cycle Start*** will begin continuous execution from where the current instruction pointer  is located on the left of the edit window. ***Single Step*** 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. The Keyboard F5 Key can be used to toggle Cycle Start/Halt.

Halt (available when GCode is running) 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 brings additional screens into view. The Tool Setup Screen is used to configure the system's parameters. Machine Axis resolution/velocity/accelerations. M Code and User Button Actions 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



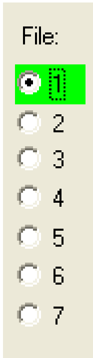
The Edit Window displays the loaded G-Code and allows editing. G-Code color syntax highlighting makes the code more readable. A right mouse click will bring up the context menu shown.

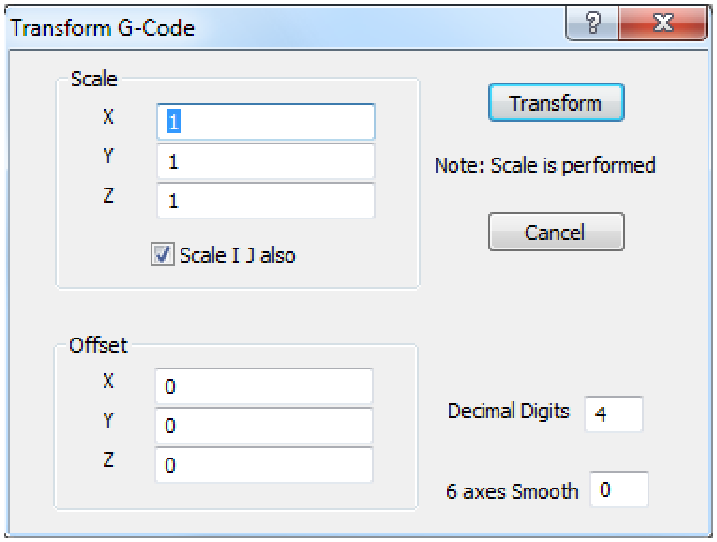
Unlimited **Undo/Redo** is supported by right mouse clicking or Ctrl-Z/Ctrl-Y hot keys.

Find and Replace is supported by right mouse clicking. Ctrl-F hot key for Find, Ctrl-H for replace.

Set Next Statement can be used to move the current point of execution to the point where the mouse was clicked. A reverse search and analysis of the preceding lines will be performed to attempt to determine an appropriate starting position and conditions for the specified line. A Safe/Resume process and movement may be performed when execution is continued. Starting in the middle of a GCode sequence can be complex. It is up to the Operator to make certain that the state and current conditions are appropriate.

Note that **Transform Sel** will bring up a utility dialog (shown below) that allows the selected G-Code to be scaled or offset. The *6 axes Smooth* option will measure an average between plotting points and allow for a smoother path. This feature is recursive, i.e. it can be repeated as many times as needed.





Toggle Block Delete "\": When highlighting several lines of Gcode and then right-mouse-clicking, users can toggle between lines that will be executed and those that will not.

Show/Hide Line numbers can be selected to display GCode Line numbers in the Left Margin.

Users can set **Make Read Only** to avoid making any further changes. Select **Allow Read/Write** to revert to writeable (Note: will be undone automatically when program restarts).

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 / Do Times

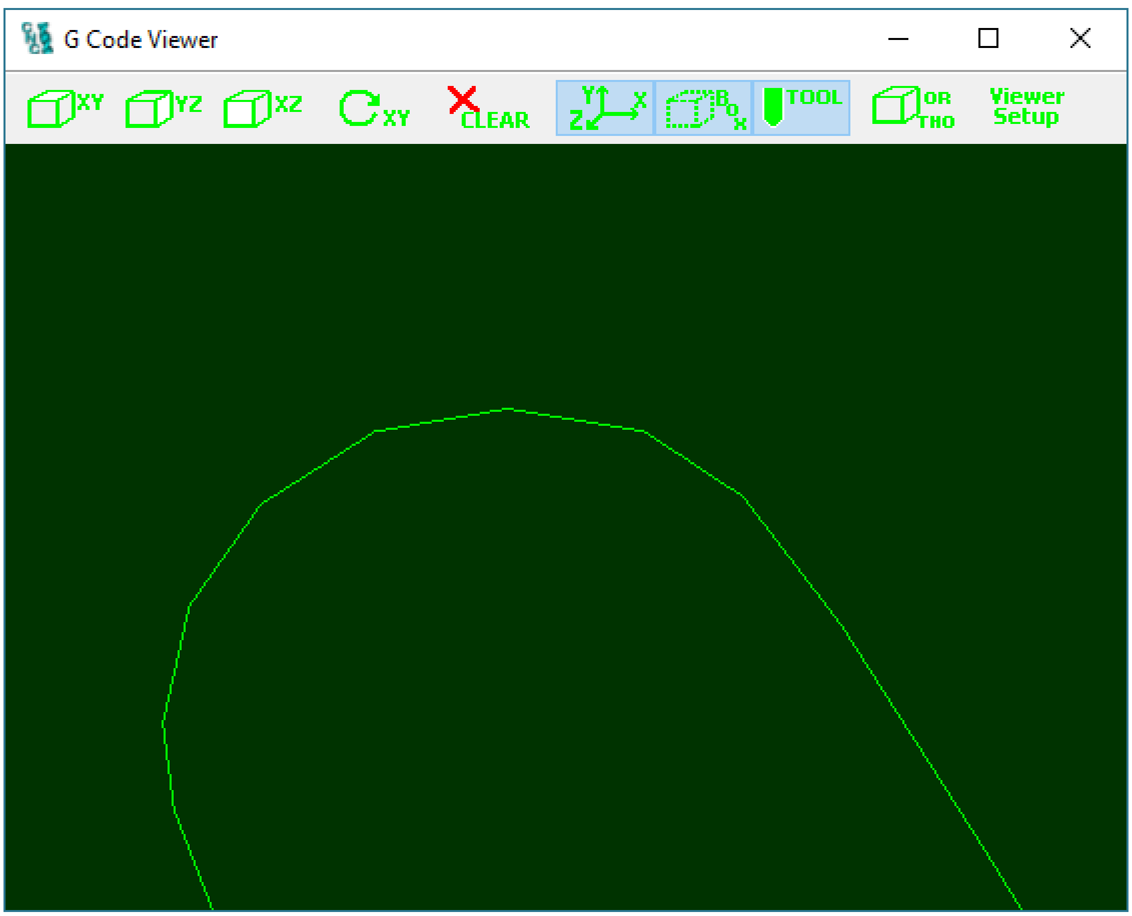
Enables Simulation Mode which allows viewing and verification of a G Code Program with or without any actual hardware connected. When Simulation mode is enabled no actual machine motion will be made. Executing or Single Stepping through a G Code program will change the Displayed Position and Plot the 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.

To perform a quick simulation for plotting or job extents verification the Run button may be used. Note that certain Jobs with manual operations, probing, etc. may not be possible to simulate.

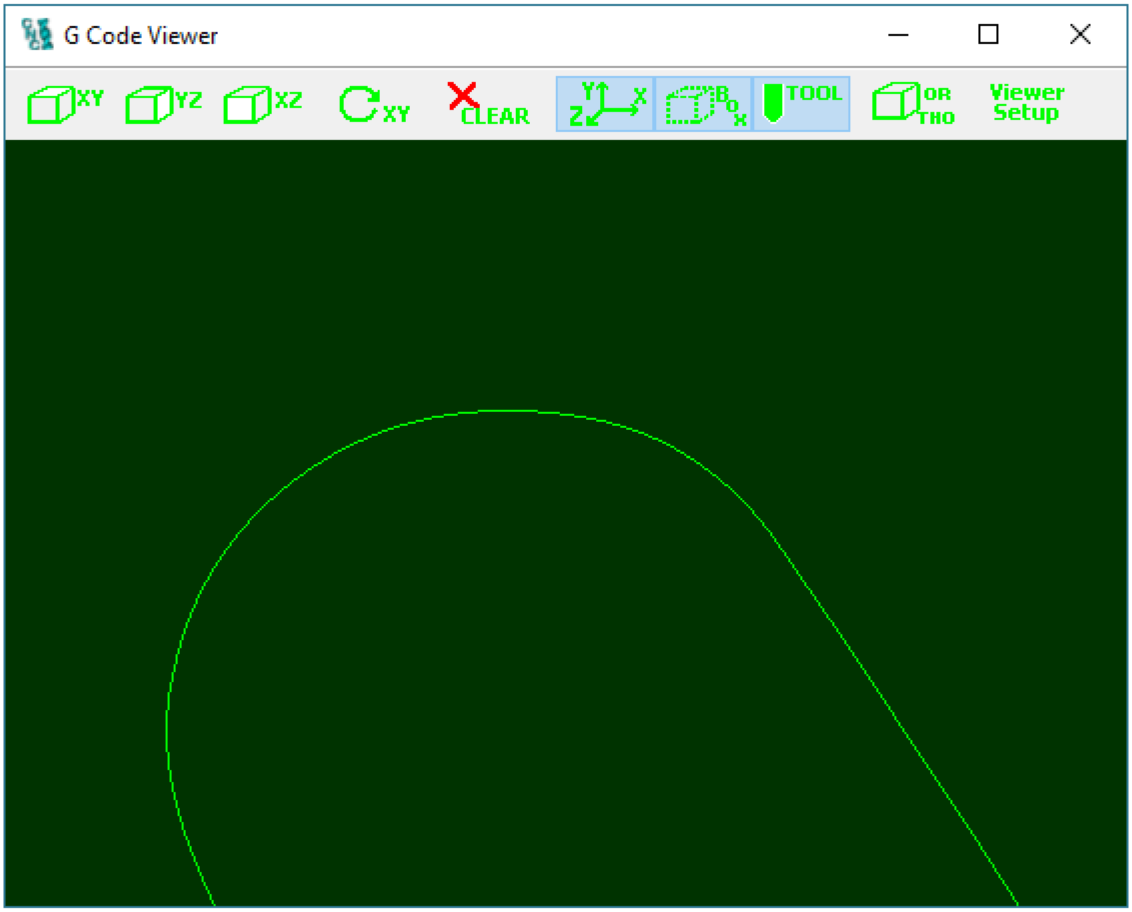
Additionaly, there is a "Do Times" Simulation option. When checked, running Simulation will fully Trajectory-plan all Paths the same as a real run (without commanding any motion) and display estimated Job Time. Displayed times have been changed from seconds to hours:minutes:seconds. GViewer Plot will then show Trajectory Planned paths rather than what raw GCode specifies.

Simulation with "Do Times" on is a few times slower than with it off, but still very fast.

Here is the GViewer zoomed in on some GCode with faceted curves (from Dynomotion.ngc top of 'Y') plotted with 'Do Times' off. Note the facets are visible when the raw GCode is plotted.



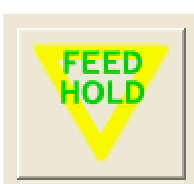
Here is the plot with 'Do Times' on. Note corner rounding has been applied for smoothing. Corner Tolerance 0.001 inches. Facet Angle 1 degree.



Block Delete

When checked, any line with a single '/' character at the beginning of the line will be deleted (skipped). When unchecked the line will be executed normally while ignoring the '/' character. The Initial state of Block Delete can be configured in the [Tool Setup File](#).

Feed Hold



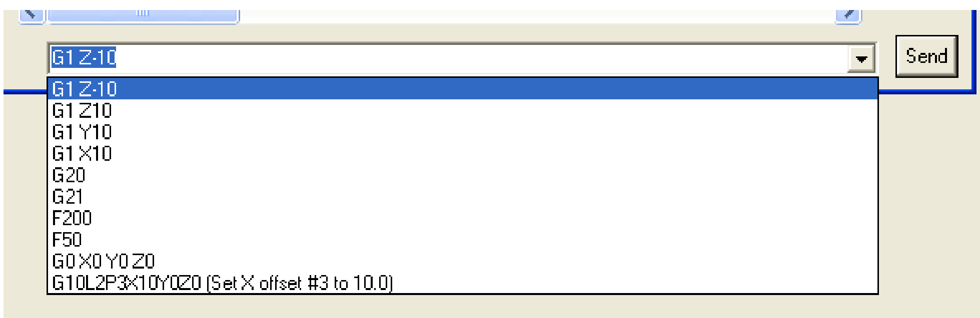
Feed Hold may be used to immediately decelerate all axes to a controlled stop. Any coordinated motion in progress will decelerate along the defined buffered path. When stopped in feed hold, the button will be shown in a toggled state. To resume motion toggle the feed hold button off. Note that while 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.

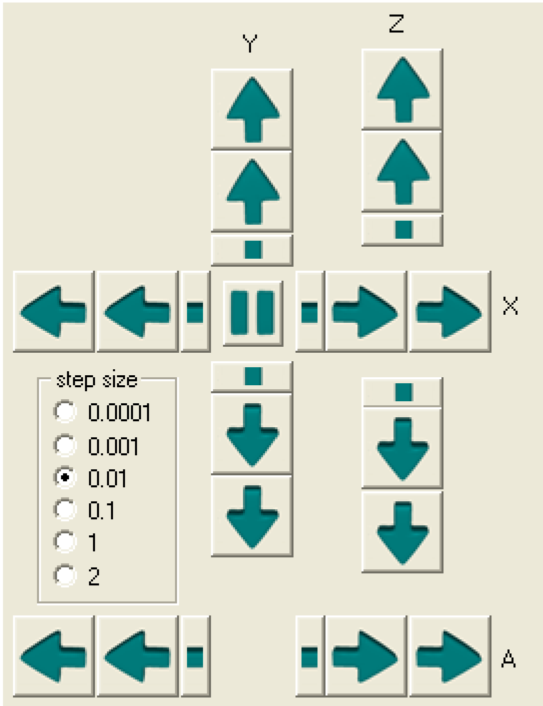
The ESC key may also be used to initiate an Emergency Stop whenever the KMotionCNC Screen has the focus.

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



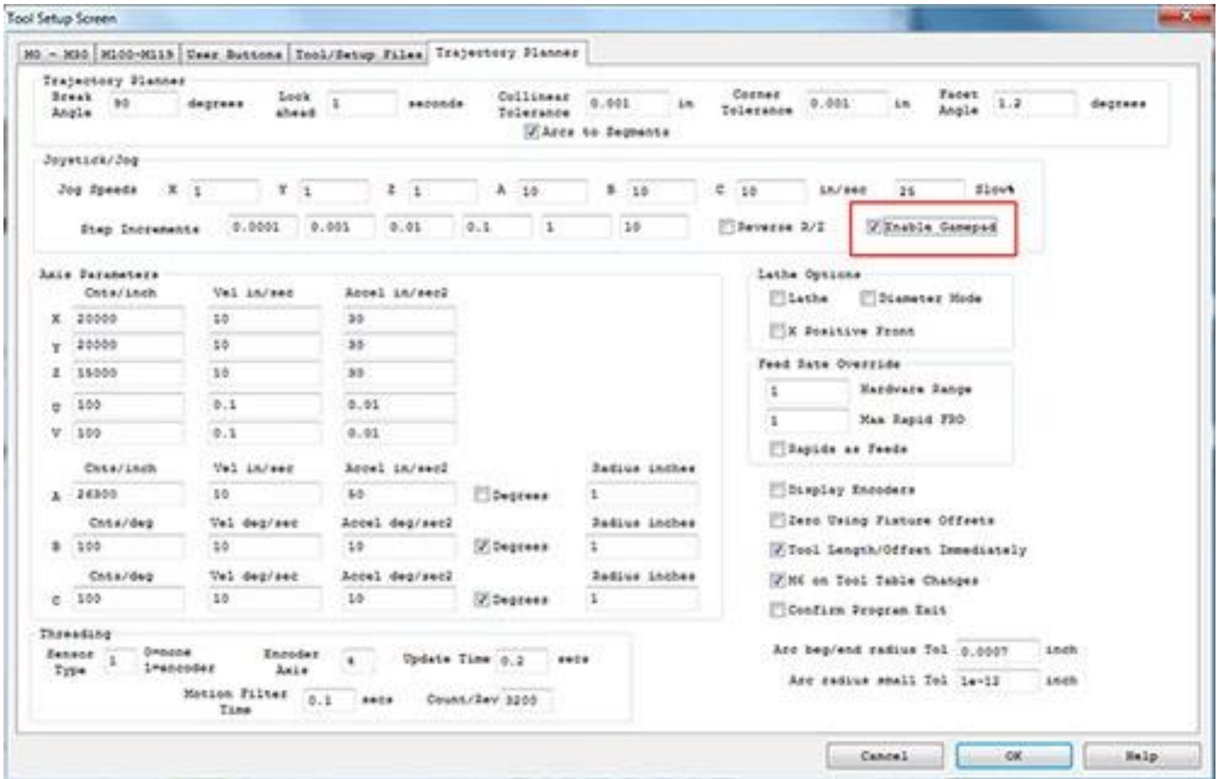
The Jog buttons may be used to move any of the axes. Pushing and holding any of the arrow buttons will cause continuous motion. There are 2 buttons in each direction for each axis. The second button moves at twice the rate as the first. The speeds for each axis may be specified in the Tool Setup Screen.

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).

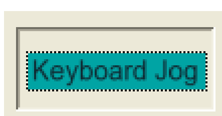
A USB Gamepad such as the one shown below or Logitech's F310 may also be used to Jog the System. Simply connect the Gamepad and it should become immediately active. The left Joystick controls x and y and the right joystick controls z and a. The same speed parameters in the Tool Setup Screen control both the Jog pushbuttons on the screen and the Gamepad.

The Jog buttons and Gamepad are also active in simulation mode

The Jog buttons and Gamepad are also active in simulation mode. Note that you must check the box next to "Enable Gamepad" (by default unchecked) in the Tool Setup Screen/Trajectory Planner:

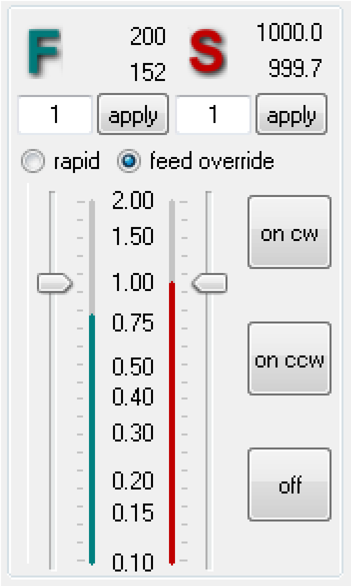


Keyboard Jog



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/Spindle Rate Override



Feed Rate Override (FRO) 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 FRO 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 FRO 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. If the FRO or (Specified Feed rate itself) doesn't seem to be having the expected result, check if the maximum velocities and accelerations are limiting.

The FRO 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 FRO 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.

Hardware Feed Rate Override (within KFLOP/Kogna) can also be used for a more instant response. However Hardware FRO may result in distorted accelerations. Users can control how FRO is performed using the Feed Rate Override Hardware Range Parameter.

Two numbers at the top of the screen show the last commanded F (Feedrate) setting and the actual instantaneous Feedrate. The actual Feedrate is also plotted as a bar graph as a ratio of actual feed rate to last programmed feed rate.

There is also an independently adjustable Rapid Rate Override, This can be displayed and adjusted by selecting the “rapid” selection. Rapid motions are performed in G Code for G0 type of motion. Rapid Rate Override always uses Hardware Override and has an instantaneous effect.

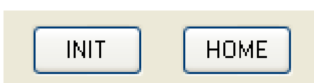
Feed Rate can be unsynchronized as distance per time for G1,G2, G3 types of motion, or may be Spindle synchronized as distance per revolution for G32 types of Synchronized Threading Motions. When synchronized threading operation is in progress the Feed Rate Icon will be displayed as:  instead of simply as  for normal operations.

Similar controls and displays exist for Spindle Control and correspond to the last specified S value in the GCode.

Constant Surface Speed (CSS) is also supported which automatically varies the Spindle Speed as a function of cutting Radius in order to maintain a constant surface speed of the cutting tool across the material. G96/G97 Codes turn on and off this mode. When operating in CSS mode the Spindle Icon will be displayed as:  instead of simply as  for normal operations.

3 buttons allow the spindle to be turned on CW, on CCW, to the last specified S setting and SSO. If Spindle Speed feedback is available in the system, the current spindle speed (in RPM) and bar graph will display the ratio of actual Spindle Speed to last programmed Spindle Speed.

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***.

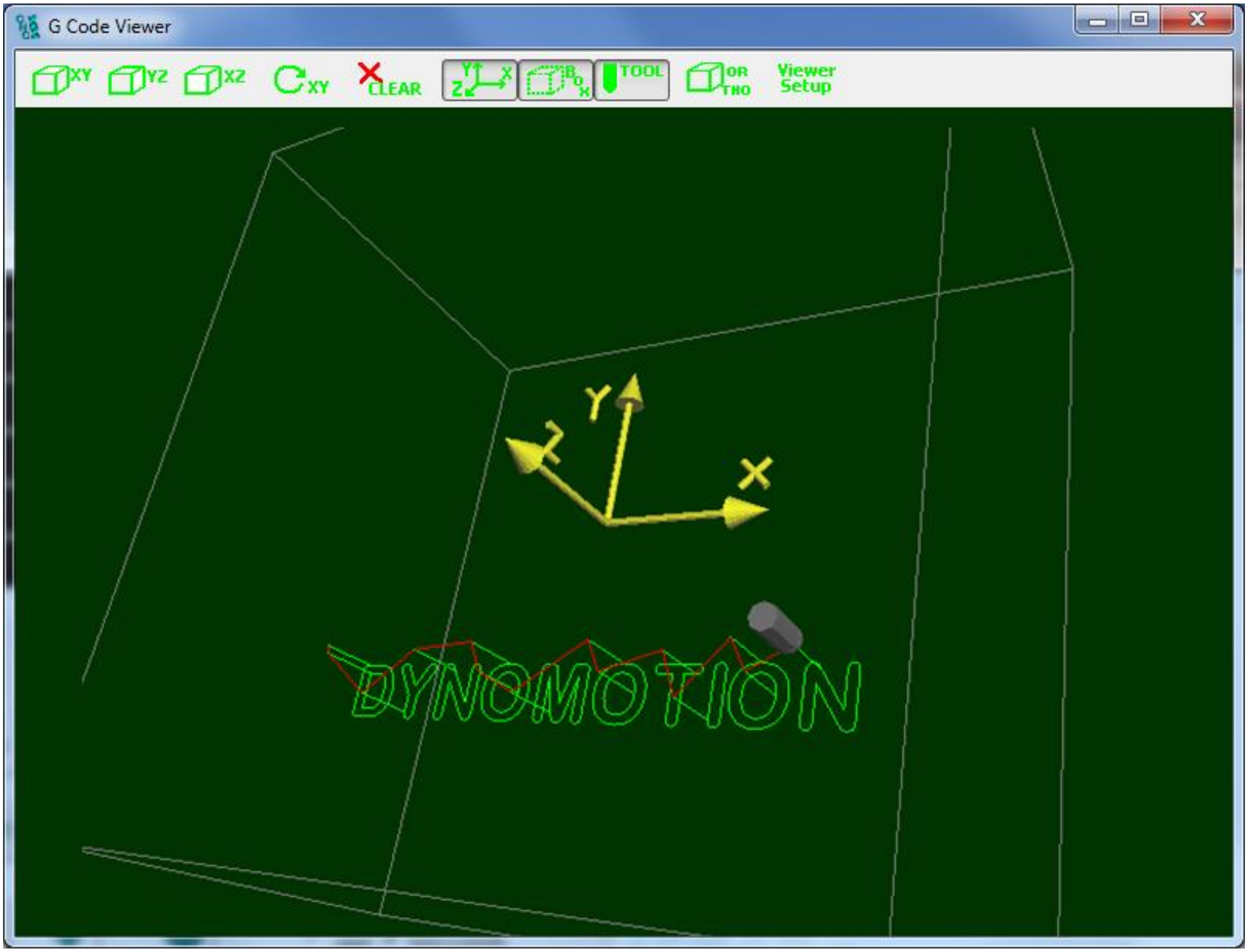
A comment in the form: (CMD,xxxxxx) will issue the command xxxxxx immediately to ***KMotion*** as soon as it is encountered by the Interpreter. Any ***KMotion*** command that does not generate a response may be used in this manner.

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 xxxxxxx.

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.

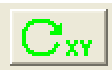
Click and drag the **left** mouse Button to translate **up/down/left/right.**

Click and drag the **right** mouse Button to translate **closer or farther.**

Click and drag **both** mouse Buttons to **rotate**.

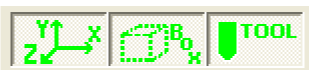
Top/Side/Front Views

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 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.

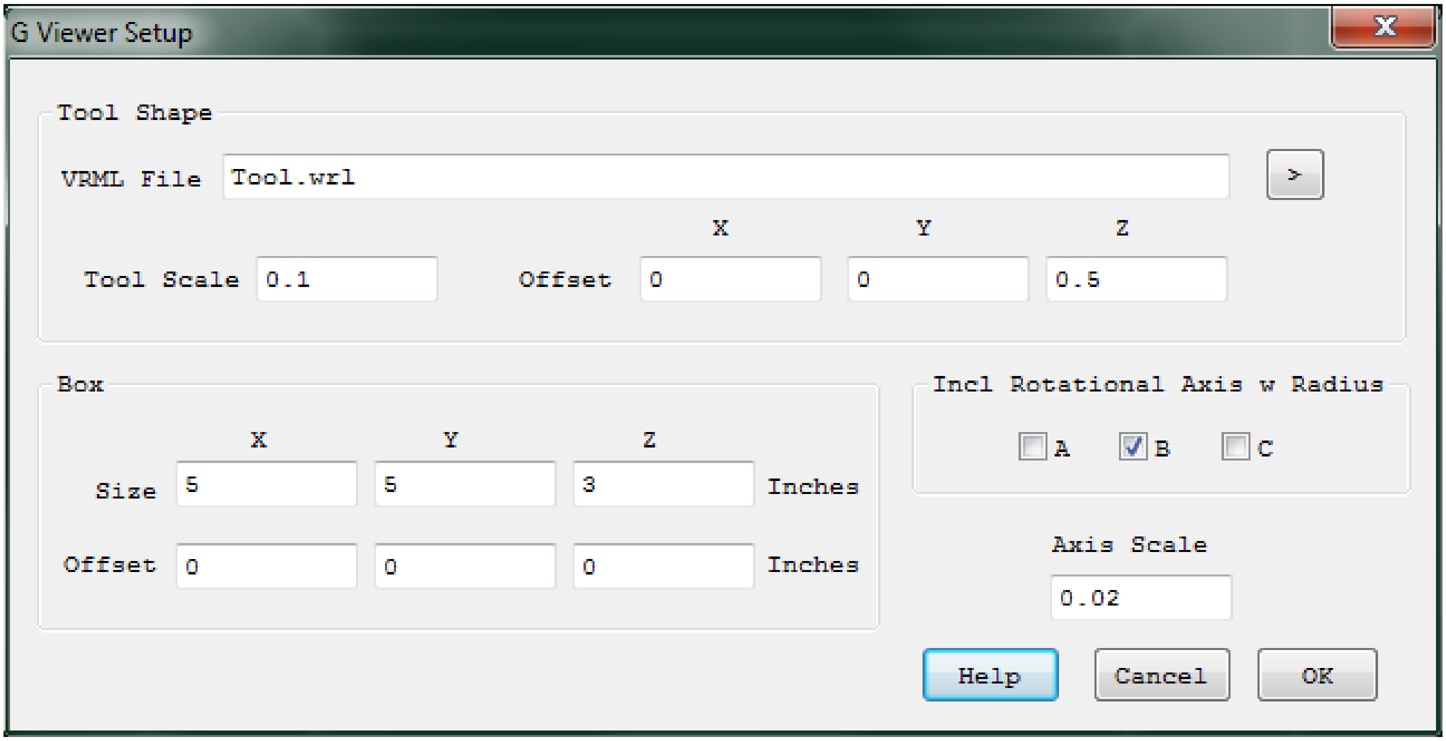


These are three latching toggle buttons which determine whether the Axis, Box, or Tool Shapes respectively are to be displayed. When latched down the item is displayed. When latched up the item is hidden. The size and shape of these items may be changed on the G Viewer Setup Screen.

This button allows users to view the screen without 3D perspective. You can toggle between orthographic and perspective view. Note how in orthographic view the position of the cursor is visible in the title bar of your window:

This button brings up the G Viewer Setup Screen which allows some customization of the G Code Viewer Screen.

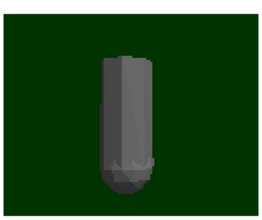
G Viewer Setup Screen



The G Viewer Setup Screen sets the imaging parameters for the G Code Viewer Screen.

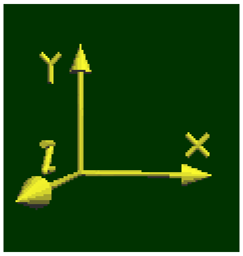
Besides the G code tools paths, the G Code Viewer Screen displays several objects, namely a Tool, a Box, and an Axis Symbol. The Tool Object and Axis Symbol are 3D VRML files that may be changed if desired. KMotionCNC comes with default files shown below.

Tool Shape



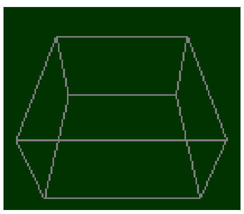
The default Tool Shape is located at: **<Install Dir>\KMotion\Data\Tool.wrl**, but maybe easily changed by entering or browsing to a new file**.** An excellent free program to create VRML files is available at http://www.artofillusion.org. The default Tool file is a 1 inch diameter sphere at the end 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. Specific Tool Shapes can be assigned to specific tools using the Tool Table.

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. When Lathe Mode is selected a different Axis Style will be used based on the X Positive Front Setting as either **<Install Dir>\KMotion\Data\AxisLathe.wrl** or **<Install Dir>\KMotion\Data\AxisLatheXFront.wrl**.

Box

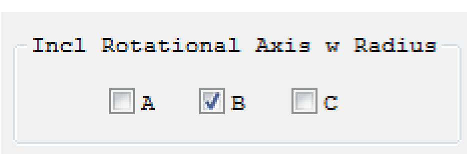


The Box serves two purposes. If enabled for display, it aids the 3D visualization and perspective of the tool paths. It is also expected to represent the working extents of the machine. When "zooming out" to one of the preset Top, Side, or Front Views, it is used to frame the view extents. Therefore 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.

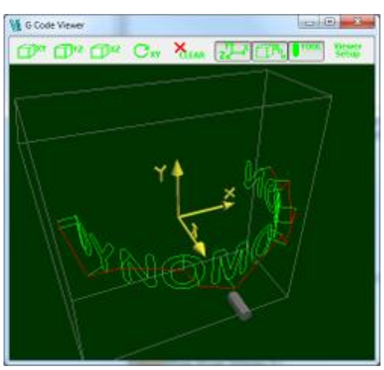
Include ABC angles

If checked, the A, B, and C angles of the tool are shown relative to the part. See video.

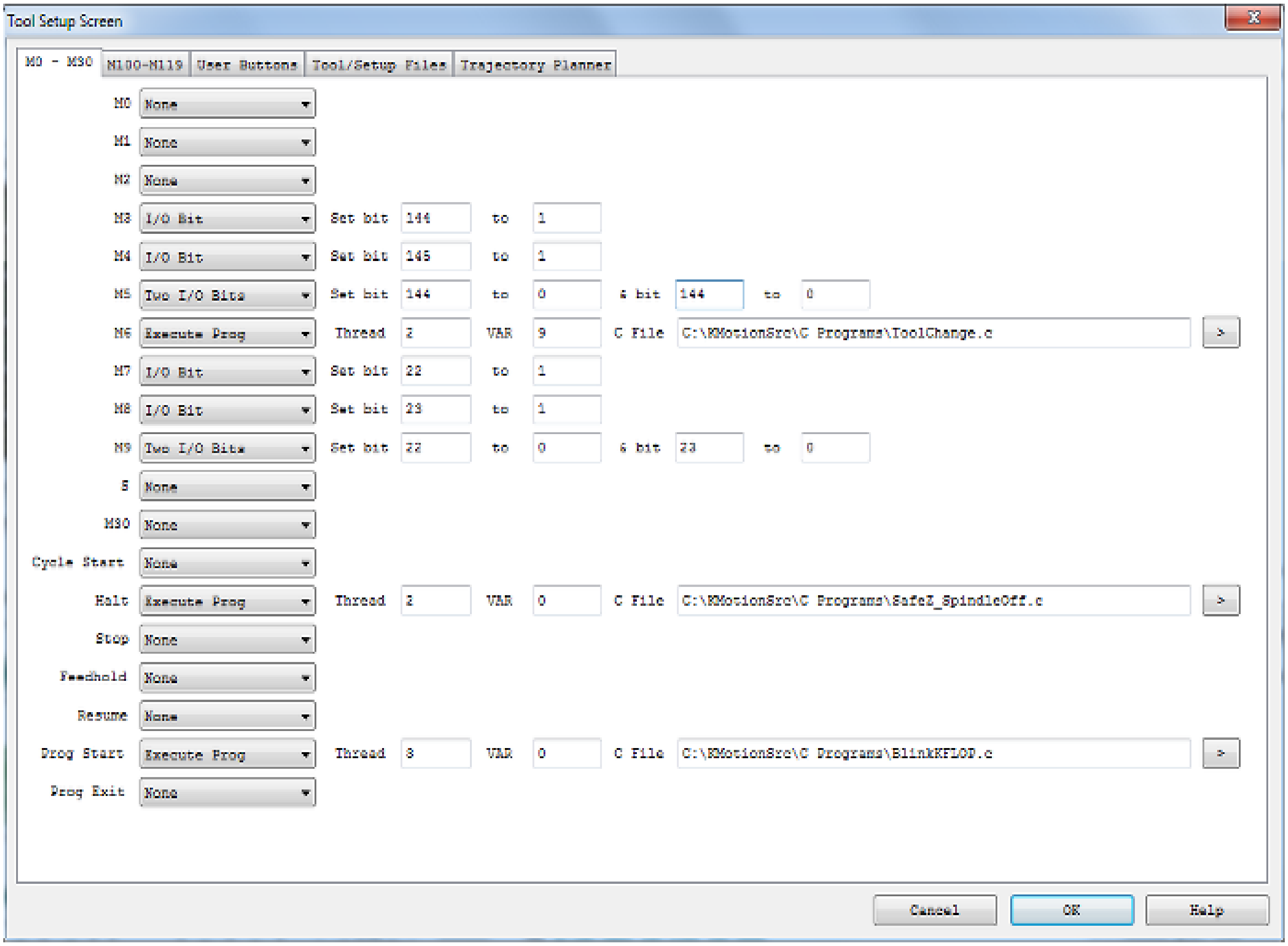
Include Rotational Axes



Includes the rotational effect of rotational axis if they represent a type of cylindrical dimension at a relatively fixed radius. The A axis rotates about the X Axis, the B axis rotates about the Y axis, and the C axis rotates about the Z axis. The example view below shows G Code plotted using B, Y , and Z Axes. The B motion is wrapped around the Y axis. The Z motion then moves away from the Y axis, and the Y motion is in the Y direction. See video.

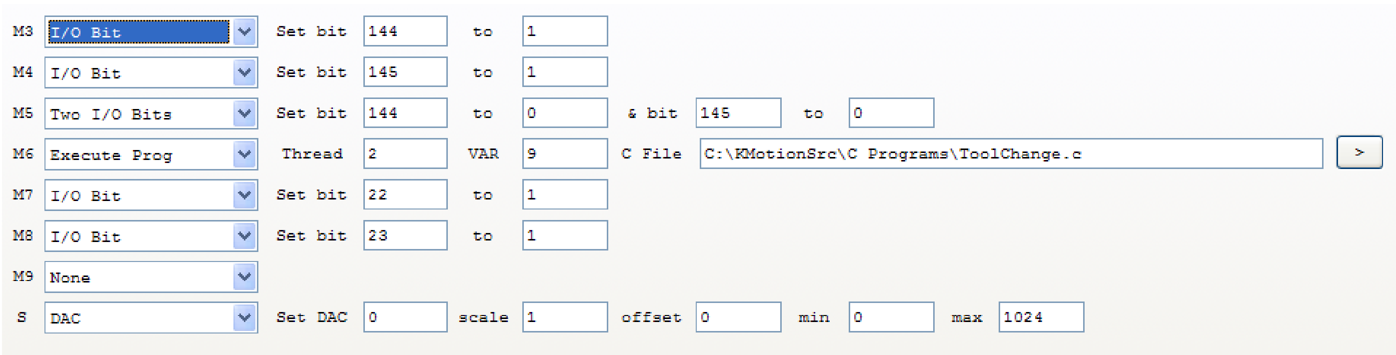


Tool Setup Screen – M0 – M30



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 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 – M0 – M30



The M Codes 0 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.

M0 - Program Stop

M1 - Optional Program Stop

M2 - Program Stop, Rewind, Reset Interpreter Settings

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)

M7 - Mist On

M8 - Flood On

M9 - Mist and Flood Off

S - Spindle Speed Setting. If C Program is specified Speed in RPM is passed to the specified KFLOP/Kogna Var as a 32-bit floating point number

M30 - Program Stop and Rewind

Additionally, Actions can be performed when certain KMotionCNC functions occur. These include:

Cycle Start - Action performed when the Cycle Start Button is pressed and before GCode begins execution

Halt - Action performed when the Halt Button is pressed after GCode stops execution. Commonly used to execute a C Program to move the Z axis to a safe height and turn off the Spindle. See the SafeZ_SpindleOff.c as an example.

Stop - Action performed after Stop Button is pressed

FeedHold - Action performed after Stop Button is pressed

Resume - Action performed after FeedHold is released

Program Start - Action performed at KMotionCNC Program Startup

Program Exit - Action before KMotionCNC Program Exits

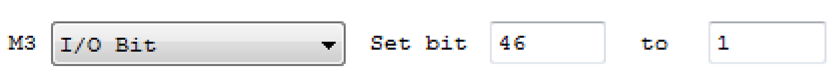
The Action that can be performed can be one of several things:

- 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)
- Wait (stall motion) until an Input to be in the specified state
- 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
- PC App Callback
- Screen Script File

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.

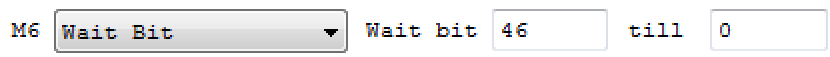
The five Action Types are described below:

For **one I/O** bit specify the I/O bit number and the state 0 or 1 to set it to.

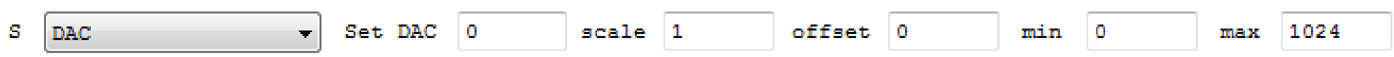


For **two I/O** bits specify the I/O bit numbers and the state 0 or 1 to set each to. Often systems with two direction spindle control will require setting two outputs that control on/off and cw/ccw. This command is designed to handle those situations.

For a special command to pause motion until an external Input is activated select **Wait Bit** and specify the I/O bit number and the state 0 or 1 to wait for.



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



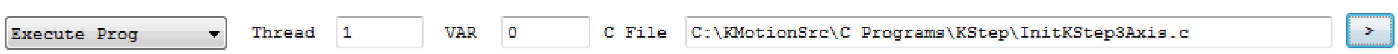
For **Execute Prog** specify the program Thread (1 through 7) where the program is to be downloaded and executed, a Persist Variable (0-199) that will be set before the program executes, and the name of the C Program to be Compiled, Downloaded, and Executed. This method is very powerful in that anything that may be programmed in C may be invoked. See the KMotion documentation and wiki (http://www.dynomotion.com/wiki/index.php?title=KFLOP_C_Programs) 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 "\KStep\InitKStep3Axis.c" s an example which completely configures all necessary parameters in the KFLOP/Kogna Board to drive 3 stepping motors using KStep's amplifiers.

Persist Variables are used for passing parameters and information to KFLOP/Kogna before executing the C Program. The parameters are Action Specific. The 'S' Action will pass the Speed as a 32-bit Floating point value. The 'M6" Tool Change Action will pass the Tool Slot and Tool ID as 32-bit integer values in consecutive Persist Variables. MCodes can pass GCode PQR Parameters. See here (http://dynomotion.com/Help/KMotionCNC/MCodesWithParams.htm) for an example. If no parameter is associated with the Action, then the Action Table Index will be passed that can be used to inform the C Program which Action invoked it. Specifying a Var value as "-1" will not write anything into any variable.

Note that normally the *.c Source Program File is specified, which simplifies making any changes as they will automatically be re-compiled and have effect. C Source programs are also independent of KFLOP/Kogna Version as they are always Compiled to the current version before use. However it is possible to specify an *.out binary file that has already been compiled for faster operation since it has already been compiled and only needs to be downloaded. When a file is compiled into binary executable code it is compiled to run in a particular Thread space (memory specific) and also for a particular KFLOP/Kogna Version. The compiler automatically marks the filename with (N).out. Where N is the Thread number that the code has been compiled to run in.

If the Filename is left blank, then it will be assumed that a program has been previously downloaded to the specified Thread and will just be re-executed. It is up to the designer to make sure a valid program has been downloaded and still exists in the specified Thread Space.

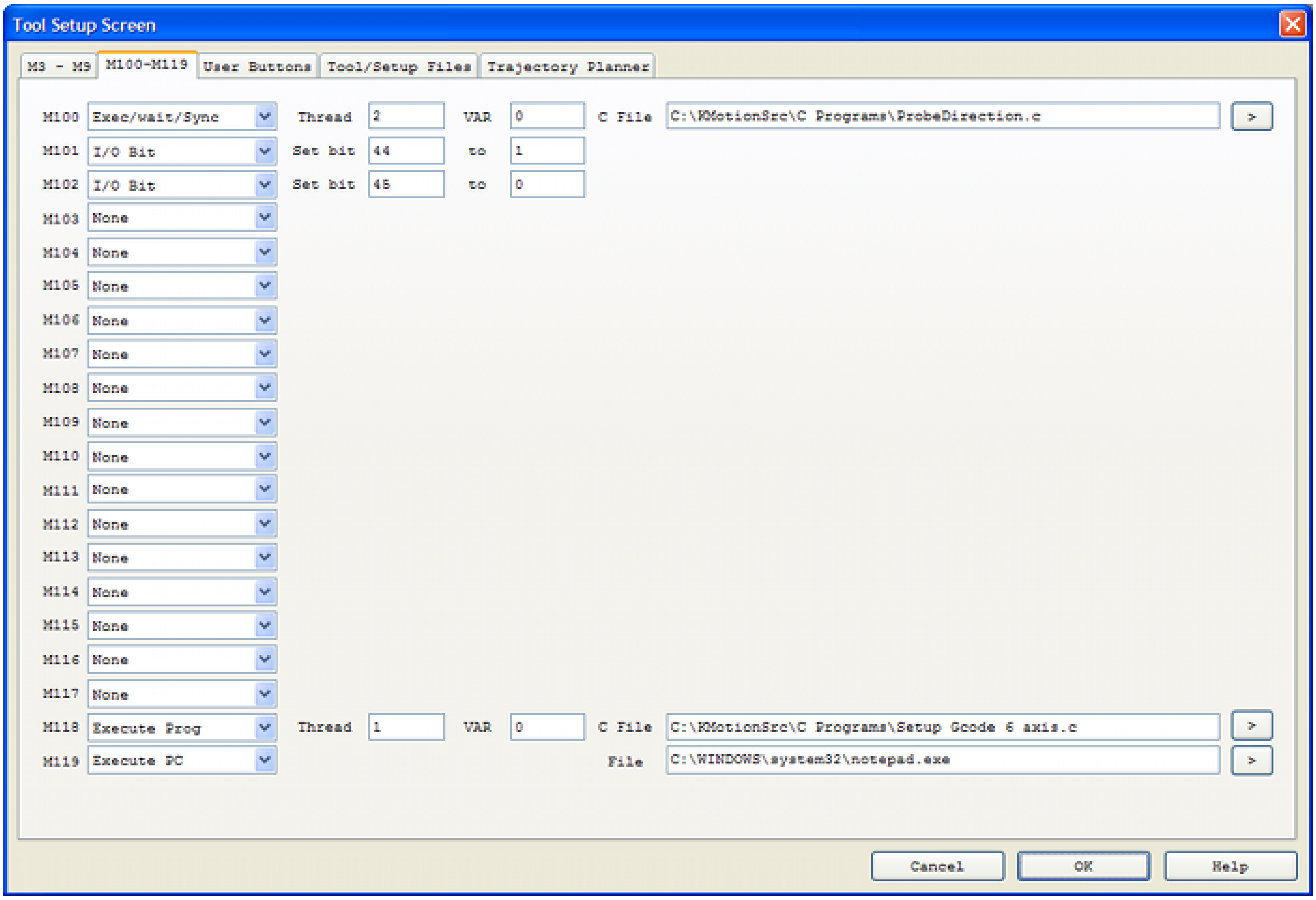
Additionally a program filename specified with a *.ngc extension will be treated as GCode and the GCode Interpreter will be invoked to execute the GCode file. Note that the GCode Interpreter is not re-entrant (Interpreter can not call itself). This means that MCodes and other Actions invoked by the Interpreter can not Invoke GCode. This restriction also applies to C Programs that may in turn execute GCodeThis essentially limits User Buttons and Special Commands (ie Program Start) to specifying GCode for execution. This functionality should be limited to simple GCode operations that should execute quickly.



PC App Callback allows a user-compiled Callback Function to be called as an Action. However, this is a feature not commonly used unless you are modifying/recompiling KMotionCNC.

Screen Script File allows users to change/select different screen script files.

Tool Setup Screen - M100 - M119

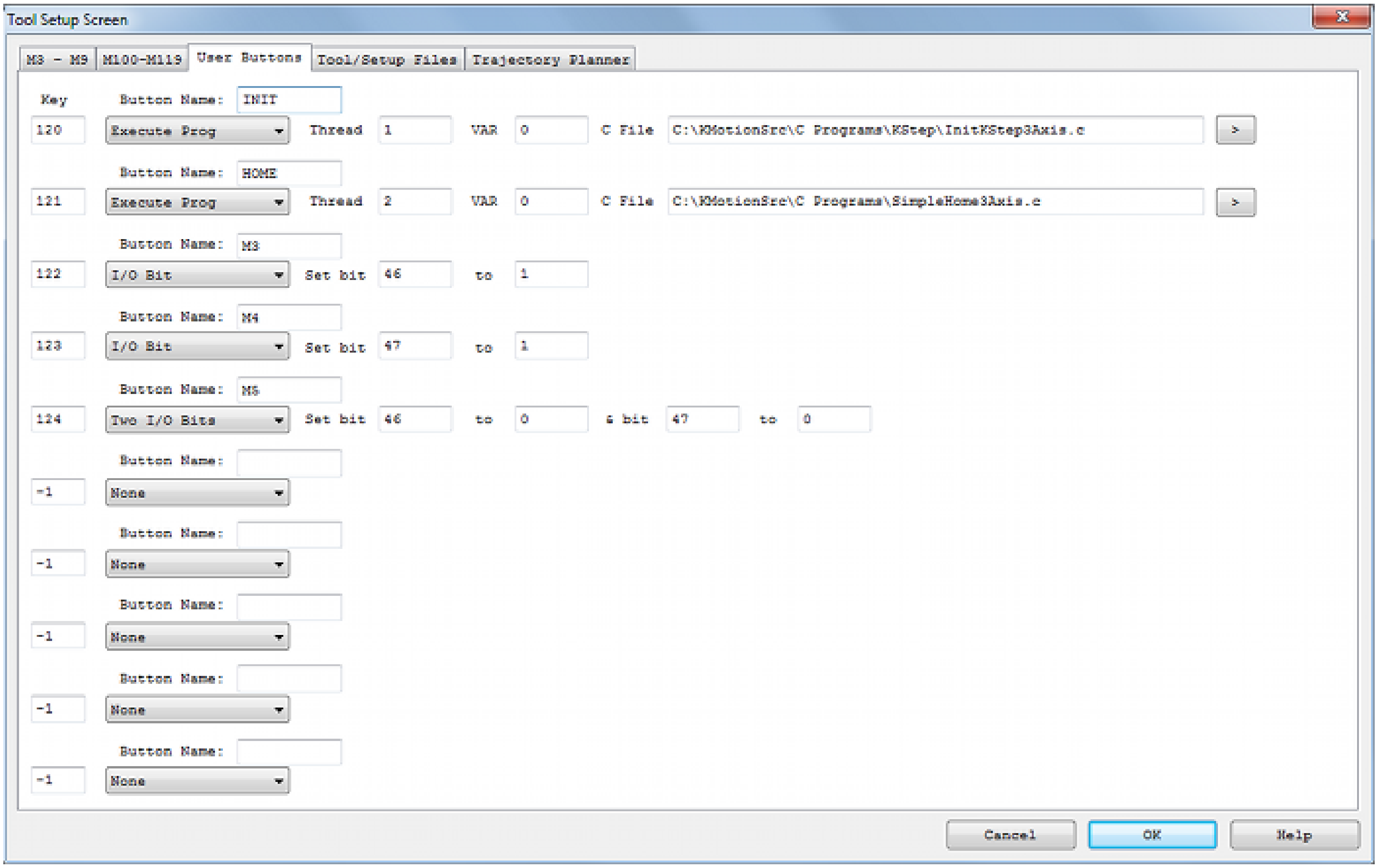


The Tool Setup Screen allows *KMotionCNC* to be configured for a particular machine tool. Each machine tool is likely to have different motion resolution, speeds, and acceleration limits. It is also likely to different I/O requirements with regard to Spindle control and such. Additionally a machine may 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

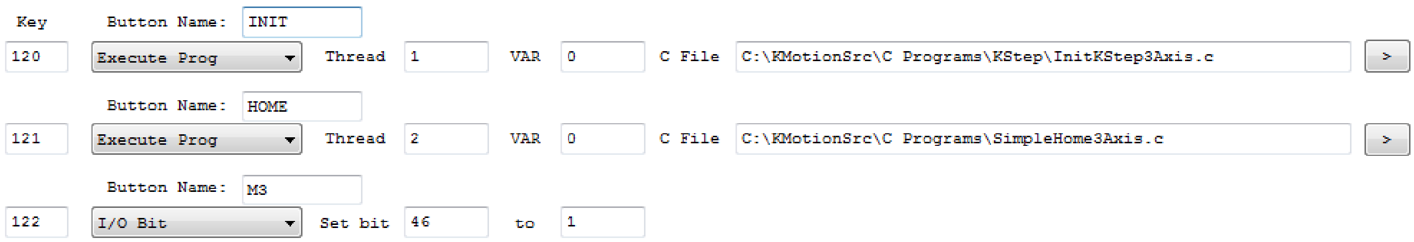
This Tool Setup Screen Tab allows the user to define up to 20 custom general purpose M codes that can be used to change IO bits, or execute User C programs, or execute Windows Programs. The action to be performed is defined in the same manner as the GCode Actions M3-M9 however parameters may also be passed from GCode to a KFLOP/Kogna User C Program. See also MCodes with Parameters.

Tool Setup Screen - User Buttons



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 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



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.

A Hot Key can also be assigned so that pushing that specific Key will trigger the Action as if the button was clicked with the mouse. The Screen must have focus for the Hot Key to work. See below for a list of numeric Virtual Key Codes:

Key Codes

Value	Description
1	Left mouse button
2	Right mouse button
3	CANCEL key
4	Middle mouse button
8	BACKSPACE key
9	TAB key
12	CLEAR key
13	ENTER key
16	SHIFT key
17	CTRL key
18	MENU key
19	PAUSE key
20	CAPS LOCK key
27	ESC key
32	SPACEBAR key
33	PAGE UP key

34	PAGE DOWN key
35	END key
36	HOME key
37	LEFT ARROW key
38	UP ARROW key
39	RIGHT ARROW key
40	DOWN ARROW key
41	SELECT key
42	PRINT SCREEN key
43	EXECUTE key
44	SNAPSHOT key
45	INS key
46	DEL key
47	HELP key
144	NUM LOCK key

KeyA Through KeyZ are the same as their ASCII equivalents: 'A' Through 'Z'

Value	Description
65	A key
66	B key
67	C key
68	D key
69	E key
70	F key
71	G key
72	H key
73	I key
74	J key
75	K key
76	L key
77	M key
78	N key
79	O key
80	P key
81	Q key
82	R key
83	S key

84	T key
85	U key
86	V key
87	W key
88	X key
89	Y key
90	Z key

Key0 Through Key9 are the same as their ASCII equivalents: '0' Through '9

Value	Description
48	0 key
49	1 key
50	2 key
51	3 key
52	4 key
53	5 key
54	6 key
55	7 key
56	8 key
57	9 key

Keys on the Numeric Keypad

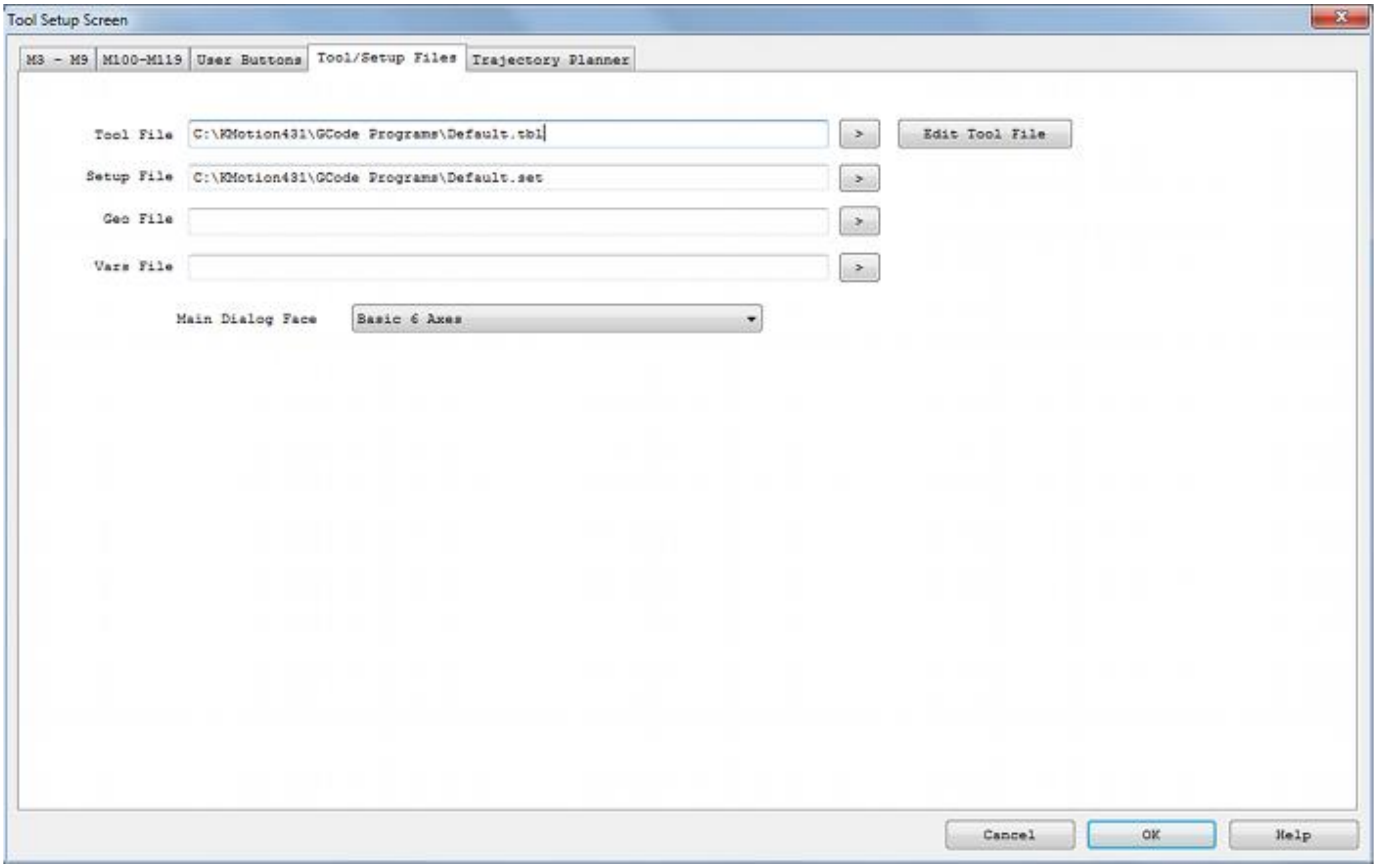
Value	Description
96	0 key
97	1 key
98	2 key
99	3 key
100	4 key
101	5 key
102	6 key
103	7 key
104	8 key
105	9 key
106	MULTIPLICATION SIGN (*) key

107	PLUS SIGN (+) key
108	ENTER (keypad) key
109	MINUS SIGN (-) key
110	DECIMAL POINT(.) key
111	DIVISION SIGN (/) key

Function Keys

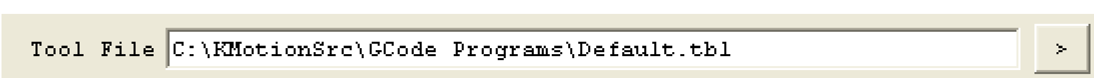
Value	Description
112	F1 key
113	F2 key
114	F3 key
115	F4 key
116	F5 key
117	F6 key
118	F7 key
119	F8 key
120	F9 key
121	F10 key
122	F11 key
123	F12 key
124	F13 key
125	F14 key
126	F15 key
127	F16 key

Tool Setup Screen - Files & Face



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 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



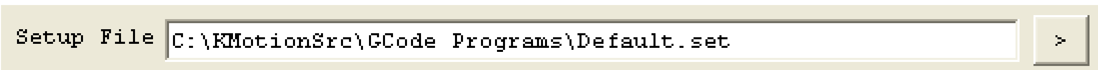
The Tool Table File specifies the disk text file which contains the table of tool definitions. In some cases the G Code Interpreter needs to know the length and diameter of the selected tool for tool path compensation. This file is used to define up to 99 tools. See also Selecting Tools.

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, CCLOCKWISE
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.

The third line defines any Table x,y offset. With an offset of 0,0 grid point row=0,col=0 will correspond to x=0, y=0. Specifying an offset will shift the table so that row=0, col=0 will correspond to the offset position. For example if the machine coordinates are such that the origin is in the middle of the range of travel, then a negative offset would be specified.

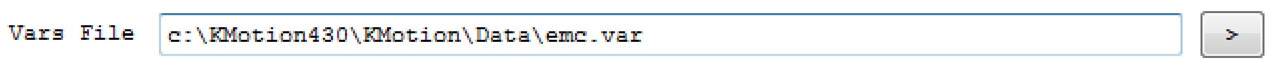
The remaining lines are row, column, x, y, z table entries.

For more information see Geo Correction Table.

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

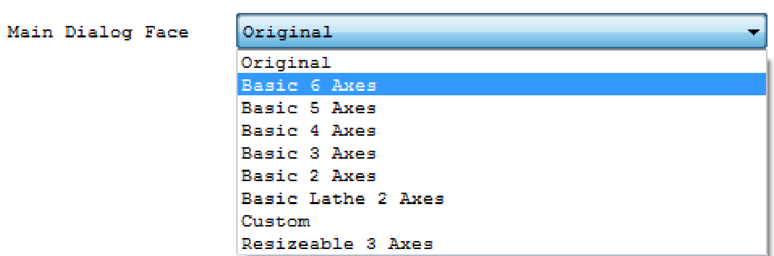
3,2,0.001747,1.008045,0.017727
3,3,0.880585,1.108997,0.059038
3,4,1.742520,1.204385,0.058162
4,0,-1.729192,1.783193,0.021364
4,1,-0.871298,1.868678,0.054451
4,2,0.005017,1.979718,0.104414
4,3,0.878944,2.104289,0.141666
4,4,1.714383,2.182679,0.144980

Vars File



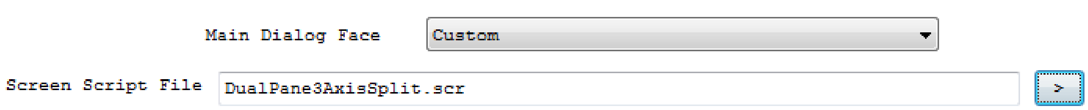
The Vars File is used to save all the GCode Variables which contain Fixture Offsets as well as other values. When Variables are saved to disk the file must already exist which lists which variables are to be saved. The file is read and all the variables present in the file will be replaced with their current values. Variables 5221-5226 contain the XYZABC coordinates of the first fixture offset. Skip 20 for each successive fixture.

Main Dialog Face

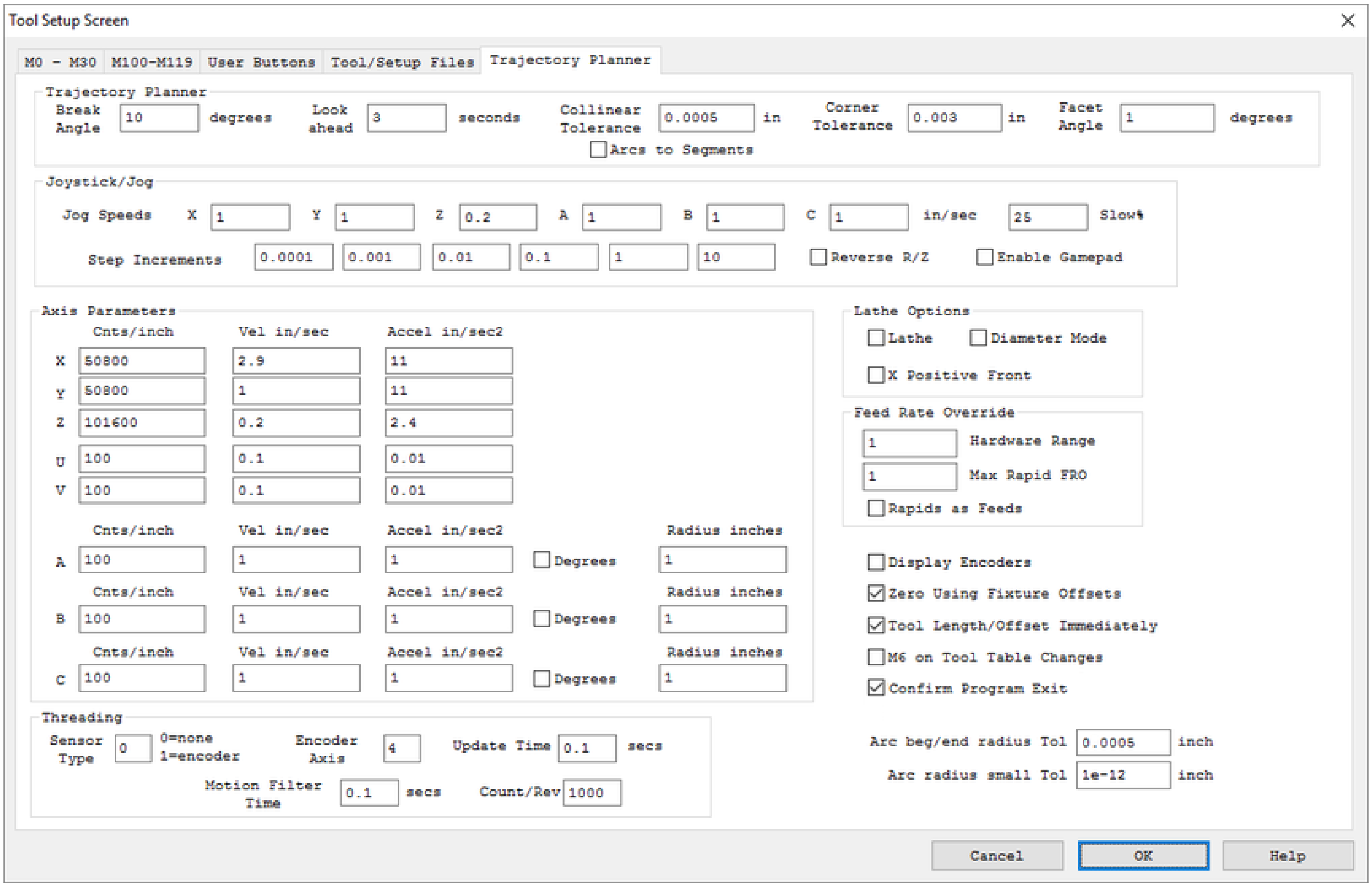


The Main Dialog Face selection selects the look of the main operation screen for KMotionCNC. Currently there are several available selections. The main difference is the number of axes, DROs and jog buttons. It is also possible to alter the dialogs by using a Windows™ Resource Editor in Microsoft Visual Studio.

By selecting “Custom”, users can load cusom screen script files which can be created and modified using the KMotionCNC Screen Editor.

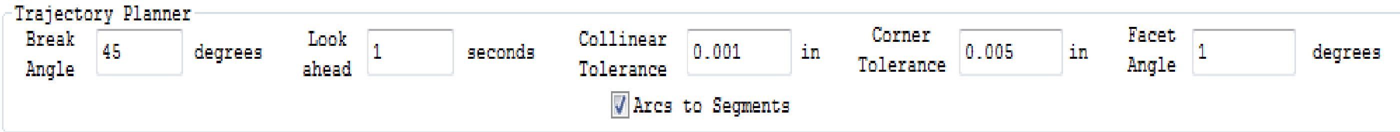


Tool Setup Screen - Trajectory Planner



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 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



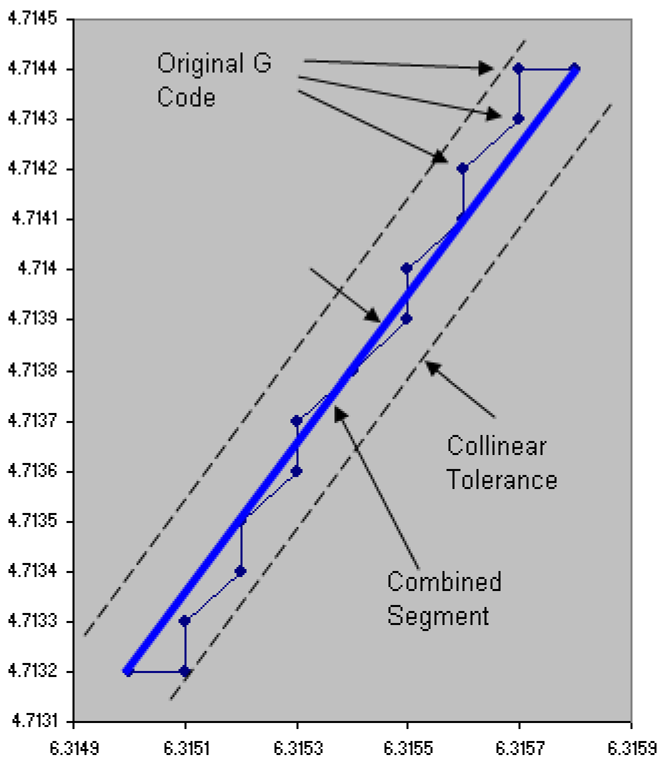
KMotionCNC contains a powerful Trajectory Planner. The Trajectory Planner utilizes a "break angle" concept to decide when a stop must be made. Vectors that are in the same direction within the "break angle" are traversed without stopping. When a "sharp" angle is detected a complete stop will be made 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
```

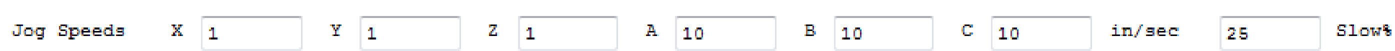


The "Arcs To Segments" option allows circular arcs that were specified in the original GCode to be replaced by a number of linear segments. Each arc will be recursively subdivided until any cord error deviation from the original path is less than the Collinear Tolerance parameter. Sub dividing the arc (actually a helix) into line segments allows the path motion to be more fully optimized by the Trajectory Planner. This is because as the motion progresses through the arc the direction changes. As the direction changes the different axes become involved and also work together in different ways (diagonal directions make use of more than one axis). This means constraints (max velocity and acceleration) vary throughout the arc. With individual line segments the Trajectory planner is able to use optimal acceleration and velocity throughout the arc. When Arcs To Segments is not selected the entire arc may be treated as a single entity and will use the most limited velocity and acceleration constraints throughout the arc for the entire arc. If the feed rate is slow and curvature based accelerations are small relative to the axis limits then there will be little performance increase in using segments.

When the "Arcs To Segments" Option is selected the Collinear Tolerance must be larger than zero. Care should be used in selecting a Collinear Tolerance so that an excessive number of segments is not generated. The number of segments increases as the square root of the inverse of the Collinear Tolerance (cutting the tolerance in half increase the number of segments by 40%).

For a description of the *Corner Rounding* feature and Corner Tolerance and Facet Angle see here.

Jog Speeds

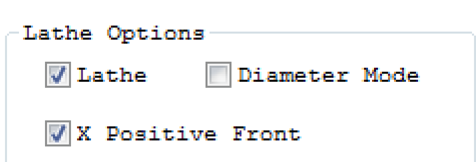


Defines the Jog Speeds for both the Jog Buttons and any attached Gamepad controller. These speeds are the maximum Jog speeds which are indicated by the outer jog arrow buttons (Jog Fast) or the GamePad joystick fully depressed. The user can define a slower speed as a percentage of the fast speed. See Also Jog Buttons.

Step Increments

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. The Reverse R/Z may be selected if the GamePad Z motion is reversed on a particular GamePad device. The *Enable Gamepad* checkbox allows you to enable or disable a gamepad. Some systems falsely indicate that they have a gamepad installed, which may cause errors. In these cases, uncheck the *Enable Gamepad* checkbox. See Also Jog Buttons.

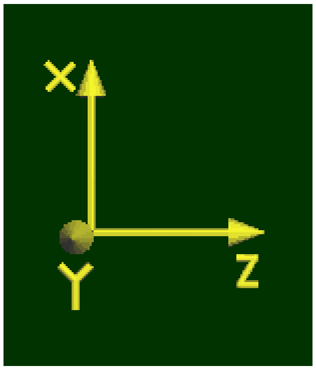
Lathe Options



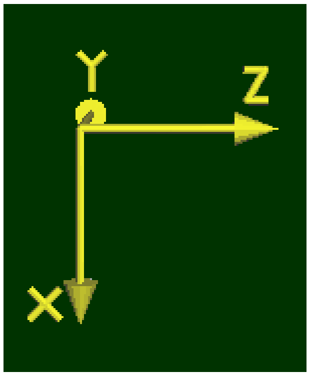
Defines options relating to Lathe systems.

Selecting the Lathe option orients the Jog Buttons and default GViewer orientations to have +Z horizontal to the right and with X vertical as most Lathe systems are configured. The "X Positive Front" rotates the axes system (and X Jog Keys) so that the tool moves toward the front of the system as X increases. This is more suitable for cutting tools mounted on the front side of the spindle. Otherwise the tool moves toward the back as X increases. This is more suitable for rear mounted tools.

X Positive Rear

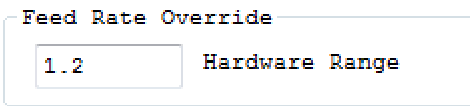


X Positive Front



When Diameter Mode is selected the Gcode X Units, X Offset Units, and X DRO Units are all in terms of diameter. In this mode actual tool cutter motion is 1/2 the specified value so the the total diameter of the cut part is the specified value. When Diameter Mode is not selected, Radius mode will be used, the tool cutter motion will be the specified value, and the cut part will have a radius of the specified value in the GCode.

Feed Rate Override



The Hardware Range value allows control over what part of the Feed Rate Override (FRO) Range is handled by Hardware, and what part is handled by Software (Trajectory Planner).

Hardware FRO has the advantage of having instant response, however it has a disadvantage of having accelerations distorted by time warping. Accelerations will be distorted by the square of the FRO setting. For example a FRO of 2.0 will result in accelerations 4X higher than planned. Hardware FRO can be thought of like playing a movie back in fast or slow motion. Another analogy might be a car that plans a trajectory accelerating from 0 to 60MPH in 10 seconds. Played at double time (FRO=2.0) would result in acceleration from 0 to 120MPH in 5 seconds! Similarly feed rates through tight curves will be proportionally increased by the FRO possibly resulting in excessive acceleration. Velocities are also increased proportionally by the Hardware FRO. This is obviously the normal intention, however speeds at or near the maximum possible for the system may now exceed the capabilities of the system. This also applies for Rapid (G0) motions. Any Hardware FRO value higher than 1.0 must have maximum system settings set in a manner that accelerations and velocities have sufficient margins to allow this increased acceleration and velocity.

Hardware FRO values less than 1.0 will be distorted in the opposite manner. Accelerations will be unnecessarily slow. In the previous car analogy played at slow motion (FRO=0.5) the car would accelerate from 0 to 30MPH in 20 seconds. Motions through tight curves will be proportionally slowed down regardless if a higher speed might be possible and closer to the desired feed rate. Hardware FRO values less than 1.0 will never cause a maximum acceleration or velocity limit to be exceeded.

Software FRO always provides the optimal motion without ever exceeding any system constraint. Maximum Accelerations and Velocities on all axes will always be honored. Accelerations to feed rate will always be optimized. Speeds through curves will always be optimal. However such optimized Trajectory Planning is complex and requires look ahead and results in some delayed response to changes. Reducing lookahead can minimize this delay.

The FRO Hardware Range Parameter sets the boundary FRO value where values below the setting will be handled by Hardware, and values above the setting will be handled by Software. Here are some examples:

Hardware Range = 0.0 will cause *all* FRO values to be handled by Software.

Hardware Range = 100.0 (a huge value) will result in *all* FRO values being handled by Hardware.

Hardware Range = 1.0 will cause all values less than 1.0 to be handled by Hardware and values greater than 1.0 to be handled in Software. This is the largest value that will never cause the set values of Max Acceleration or Max Velocity set in the Trajectory Planner to be exceeded.

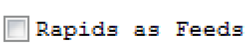
Hardware range = 1.2 will cause all values less than 1.2 to be handled by Hardware and values greater than 1.2 to be handled in Software. This setting is useful if it is required to have minor increases (+20%) and all decreases have an instant effect. However this requires that the set values of Max Acceleration or Max Velocity set in the Trajectory Planner be set by a factor of $1/1.2^2 = 0.69$ of the system capabilities in order to provide margin to allow for the Hardware FRO.

Hardware Range = 0.5 will cause all values less than 0.5 to be handled by Hardware, and values greater than 0.5 to be handled in Software. In most cases this will result in a similar result as Hardware Range = 0.0 (with all changes handled by Software). However in cases for example where the FRO is suddenly reduced from 1.0 to 0.25 the system will first reduce the FRO to 0.5 in Software, and then the remainder, another factor of 0.5 in Hardware. The Software effect will be delayed but the hardware effect will be instant. This provides most all of the benefits of Software FRO in the normal ranges of (ie 0.9 - 1.1) while also allowing a means of instantly slowing down.

Max Rapid FRO

Limits the Rapid FRO to the specified value. Rapid FRO is always performed in hardware so values Rapid FRO values greater than 1.0 will exceed the specified Velocity, Acceleration, and Jerk specified in KFLOP. Use this limit to avoid FRO values that would cause Velocities, Accelerations, or Jerks that the system would be incapable of performing. This value is commonly set to 1.0 if the parameters specified in KFLOP are set at the maximum possible. To allow FRO values greater than 1.0 the system must have sufficient margin to be able to exceed the specified parameters. Values less than 1.0 are not commonly specified.

Rapids as Feeds



Selects whether Rapids are performed as 3rd-order (Jerk limited) motions or 2nd order Feed Motions. Most systems will not select this option so that Rapids are performed as faster/smoother 3rd-order Jerk-Limited motions. However KFLOP performs 3rd order Rapids as a single linear interpolated multi-axis motion at the actuator level. This is fine for normal Cartesian xyz linear systems however may cause problems for highly non-linear kinematic systems as a straight line in actuator space is not likely to be a straight line in real CAD space. For example a Delta Platform is an example of a non-linear system. This may result in a rapid motion path from one point to another to be along a somewhat unpredictable curved path. In some cases this may result in an undesirable crash. [Geometric Correction] is also a form of nonlinearity. However Geometric correction is usually not sufficiently non-linear to cause problems unless long rapids are performed passing very close to part surfaces or features. If nonlinearity is an issue for your system, select this option and Rapids will always be performed along a straight line as they will be subdivided into small segments where proper actuator positions are calculated to keep the motion along a straight line. Regardless of whether this option is selected or not the Velocity and Acceleration will be set by the KFLOP Axis Parameters not by the current GCode Feed Rate or Trajectory Planner Settings.

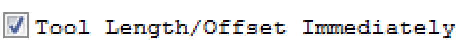
Display Encoders

Displays in the DROs the Measured Encoder Position for axes that are configured with an Input Mode other than None. When Un-checked all DROs display the commanded position.

Zero Using Fixture Offsets

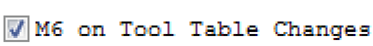
Zero/Set Buttons near DROs allow the User to Set the DRO to Zero or a Specified Value. This is accomplished by adjusting GCode Offsets. When this option is selected then the currently selected Fixture Offset will be adjusted. When unchecked the Global G92/G52 Offset will be adjusted.

Tool Length/Offset Immediately



This option causes the KMotionCNC Drop Down Tool Selector to immediately select and apply the Tool Length (and offsets) Compensation without having to select and turn on the compensation in GCode with "HxxxxG43".

M6 on Tool Table Changes

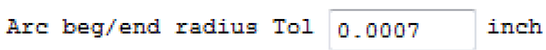


This option causes the automatic execution of an TxxxM6 operation after the Tool Table has been edited to re-select the specified Tool so the selected Tool ID, slot, offsets, image all reflect the current state of the Tool Table.

Confirm Program Exit

By default checked, this option prompts a confirmation that the user wants to exit KMotionCNC.

Arc beg/end radius Tol

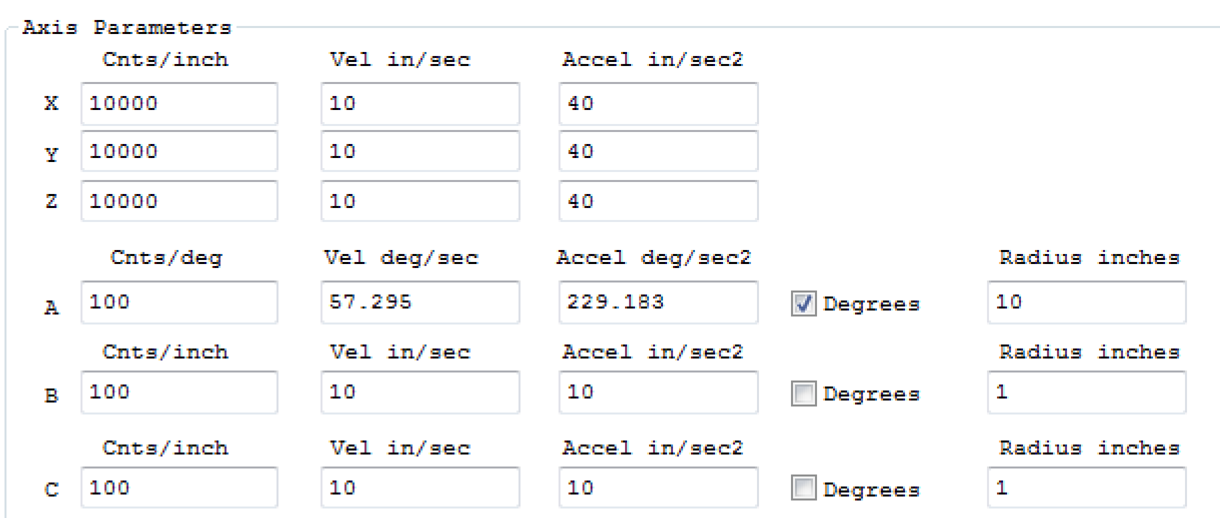


The GCode indirectly specifies the Start of the Arc (previous position), the End of the Arc (Specified Position), and the Arc Center Point (Previous Position + IJK offset). The distance between the Start to the Center and the End to the Center should be the same. Specifies the allowed Tolerance the GCode Interpreter will allow between the beginning and ending radius of an arc. This should be a very small number within the rounding tolerance of the numbers.

Arc radius small Tol

Tolerance parameter for allowing Arc Radius being smaller than half the distance from the start of Arc to End of Arc when specifying Arcs using R mode.

Axis Motion Parameters



The Axis Motion Parameters define the scale, maximum feed velocities, and maximum feed accelerations for each of the six axis.

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/sec^2. 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 all axes from one point to another along a straight line as quickly as possible without violating any axis involved in the motion constraints. To change the speed of G0 Rapid motions change the axes configuration settings in the motion controller.

Axes A,B,C sometimes are angular axes and are programmed to move angles of degrees rather than linear distances. If so select "Degrees" so that the axis moves in degrees regardless of the Inch or mm modes in GCode.

The best method for performing coordinated linear and angular combined motion is to use G93 Inverse Time Mode. This mode allows the CAD system to generate GCode that will cause the feed rate at the tool to always be at the desired feed rate based on the geometry of the system. However this requires a time (specified by its inverse) for each GCode block to be computed and included in the data stream. If Inverse TIme GCode data is not available a Radius for each Angular Axis can be specified.   This allows the Trajectory planner to convert angular motion rates to linear motion rates. This assumes that the tool will remain at a relatively constant radius from the rotation axis (ie. engraving on a cylinder). Whenever Degrees is specified with a non-zero radius then the Trajectory Planner will consider the motion for that axis to be at the equivalent radius and orthogonal to all other combined motions.

As an example for the settings shown above for Axis A the maximum linear Velocity and Acceleration rates are the same as X, Y, and Z assuming a 10 inch radius about the A Axis of rotation.

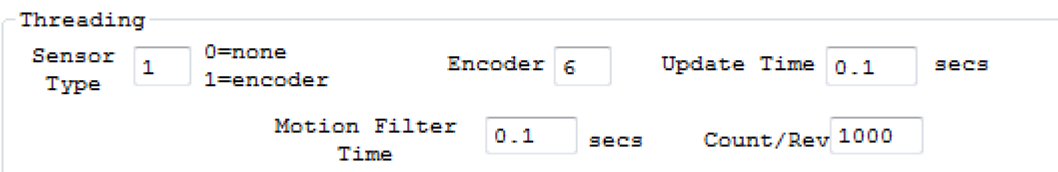
57.295 deg/sec x π/180deg x 10in = 10in/sec

229.183 deg/sec^2 x π/180deg x 10in = 40in/sec^2

The GCode sequence shown below will perform a G1 Feed a linear distance of 10 inches in both X and A. Because the X and A axis are assumed to be orthogonal the combined distance will be assumed to be 14.1 inches total. At a feed rate of 60 ipm = 1 ips, the total Feed Time will be 14.1 sec (plus a small acceleration time).

G20
G0 X0 Y0 Z0 A0
G1 X10 A57.295 F60
M2

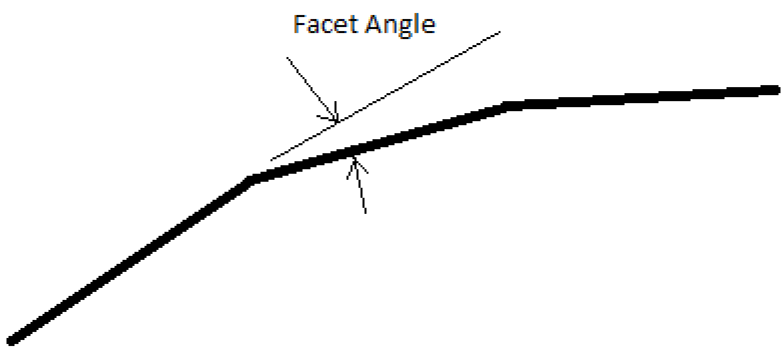
Threading/Spindle Settings

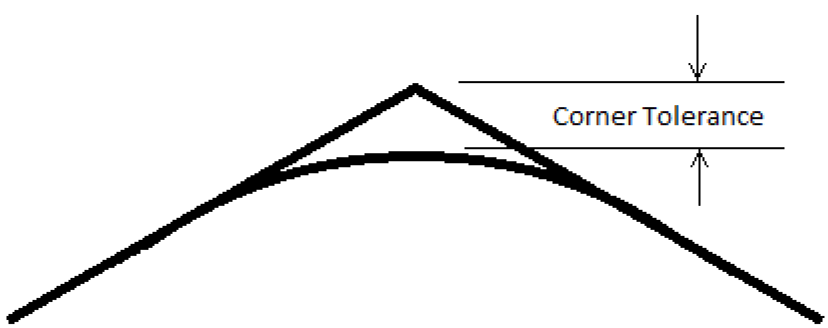


Spindle Speed measurement is supported as well as Single Point Threading. A quadrature encoder is required on the spindle. Specify the Sensor Type as 1 to enable the Spindle measurement. Configure the Axis Channel that is configured to read the encoder (Note this is not the encoder channel, rather it 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.

Trajectory Planner Corner Rounding Settings

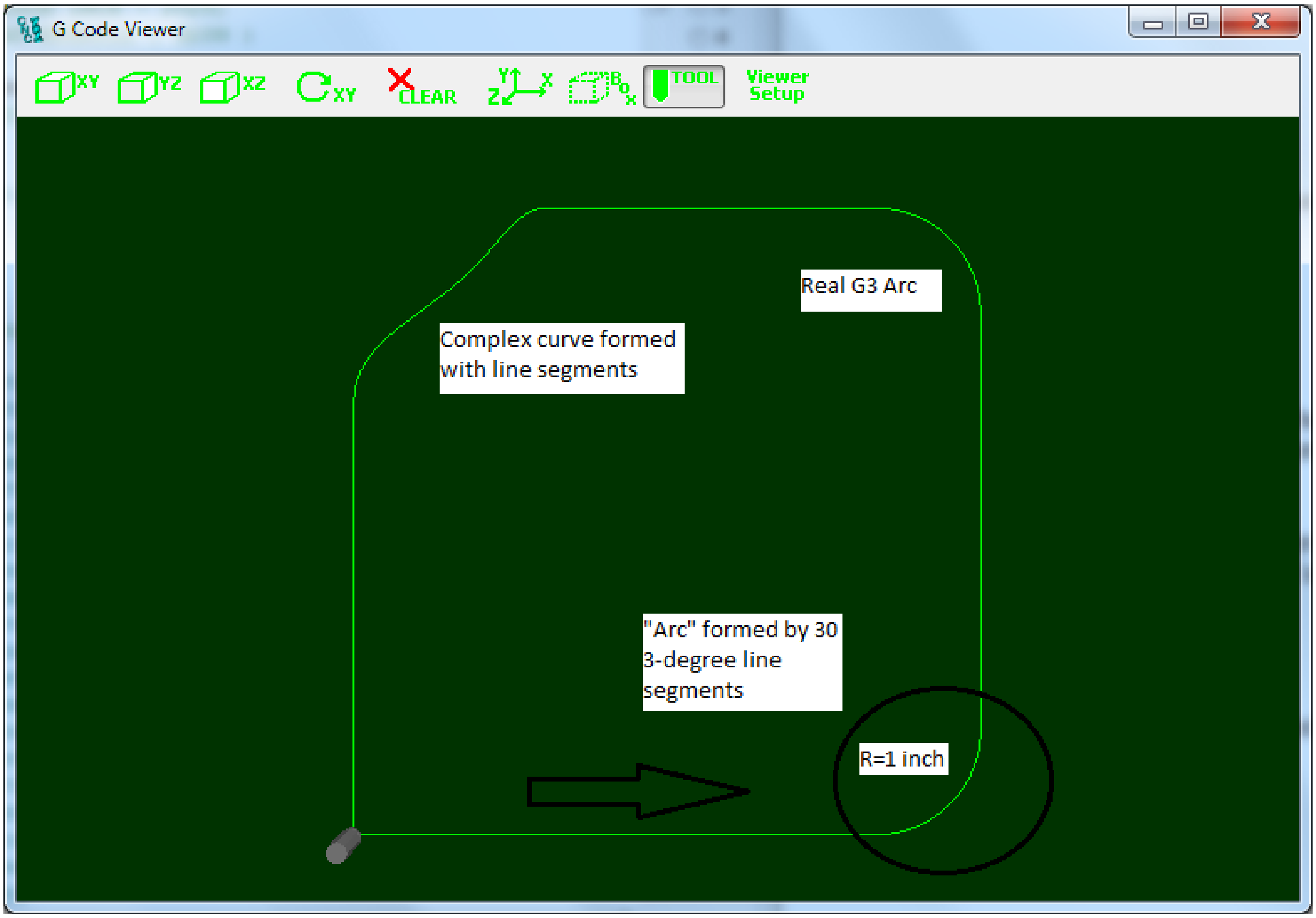
The KMotionCNC Trajectory planner contains two parameters (Facet Angle and Corner Tolerance) that can be used to smooth paths generated by line segments in the GCode. Although shown below as 2D paths this smoothing applies for 3D paths as well. Standard GCode and many CAD systems do not support arcs 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/sec^2 acceleration in both X and Y. The Gcode used can be downloaded here.

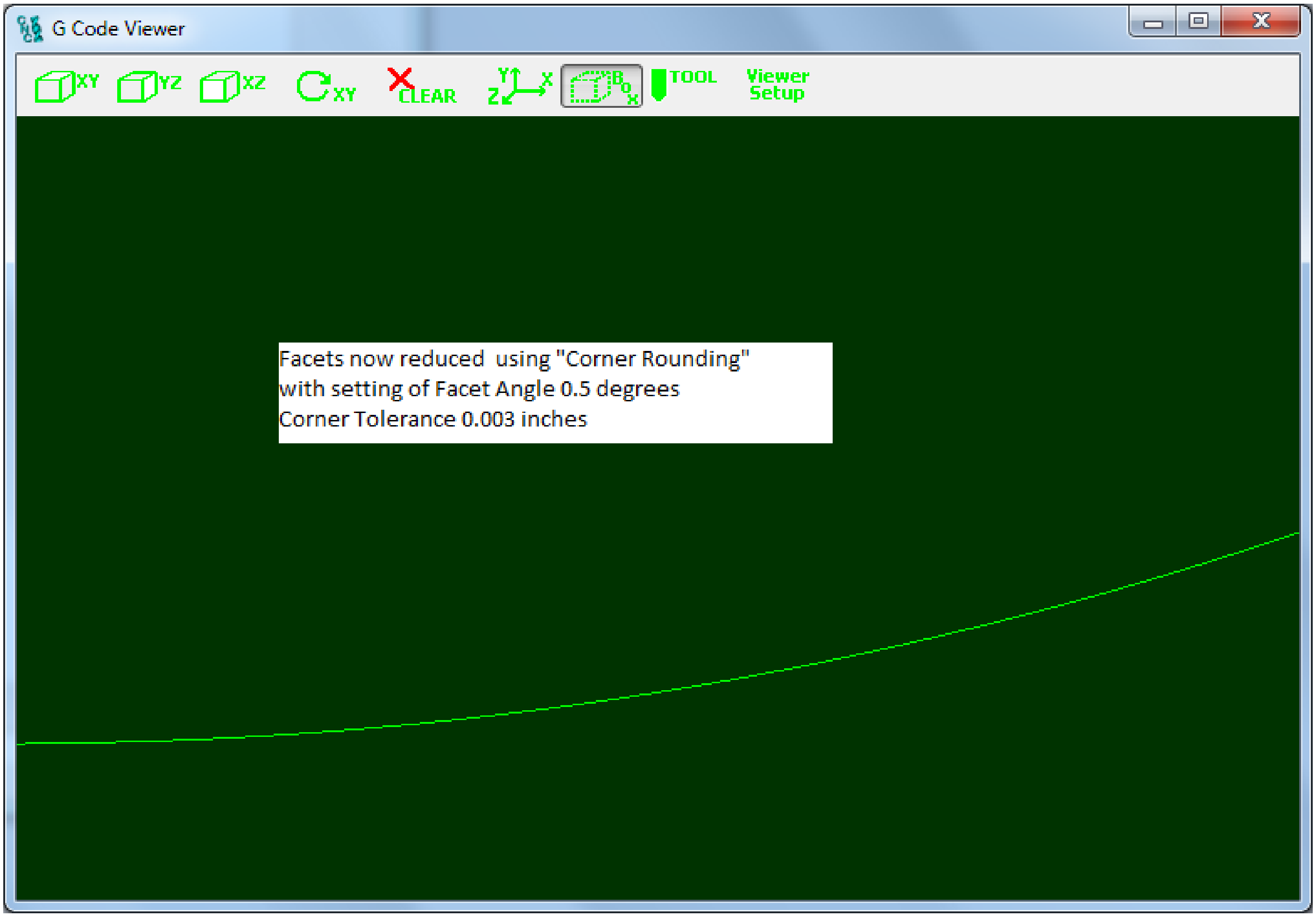
Here is the path Overview.



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.


G Code Viewer
XY
YZ
XZ
XY
CLEAR
Y
X
Z
TOOL
Viewer
Setup
CAD generated 3-degree angle facets, ~0.05 inches
(1.3mm) just barely visible

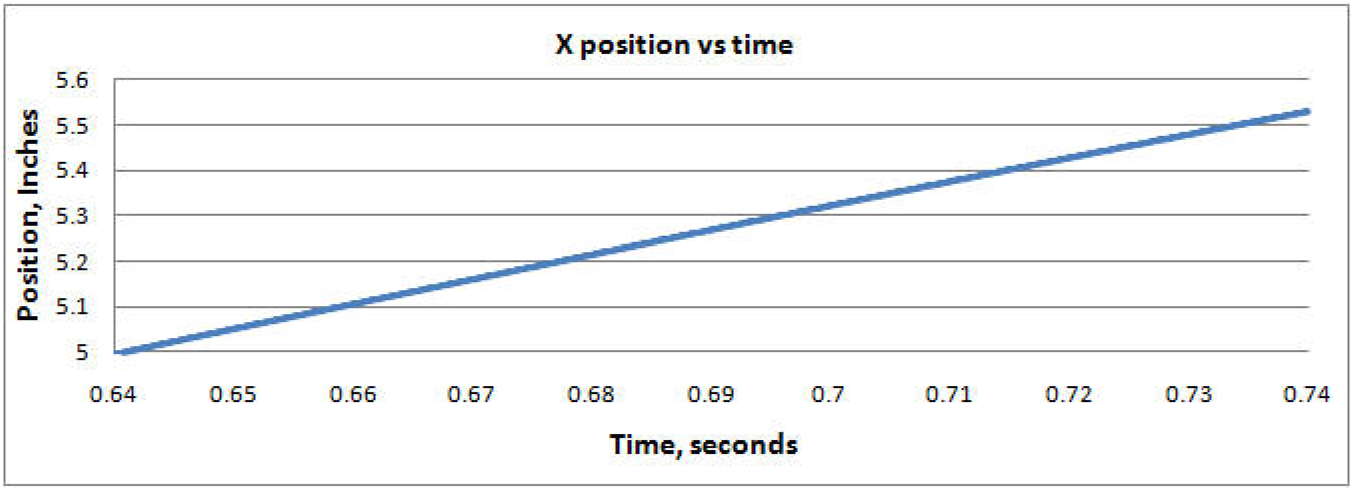
In the plot below the Corner Rounding is enabled and the facets are now reduced in size and form 0.5 degree angles that can now no longer be seen.

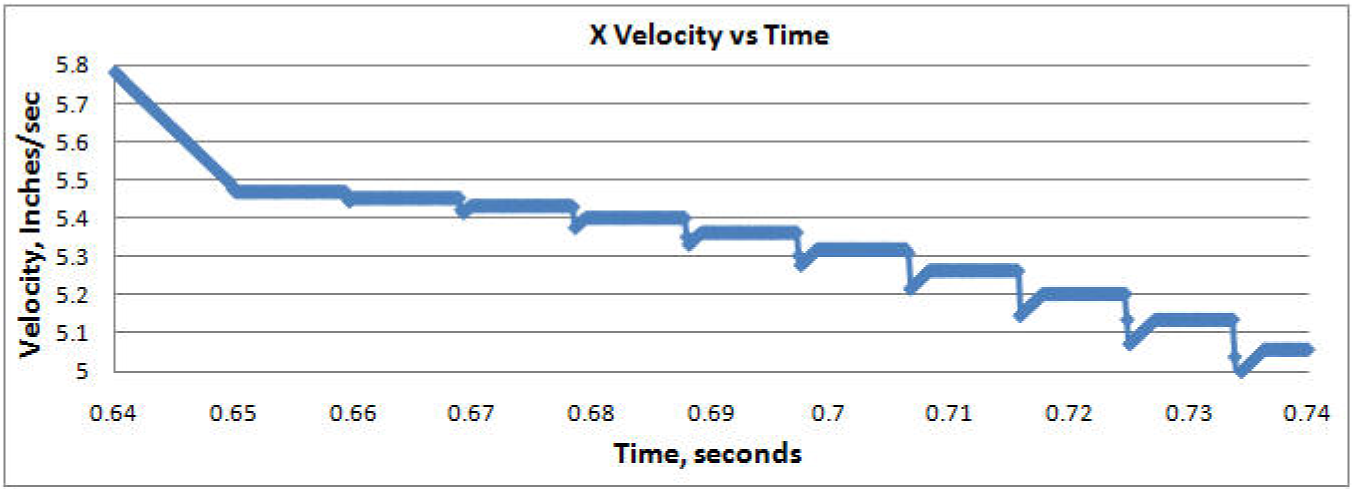


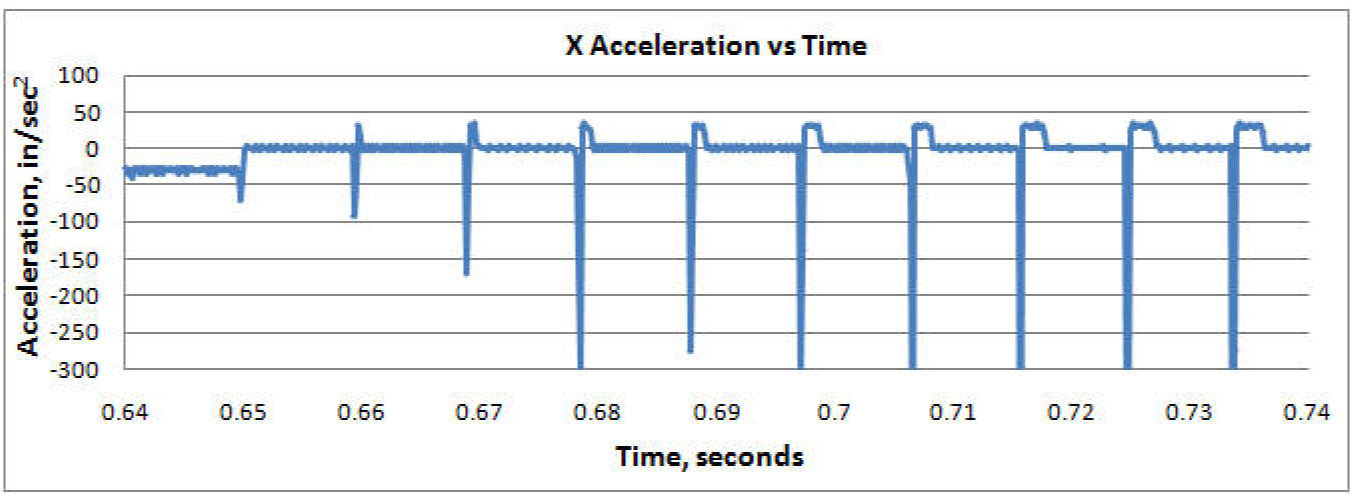
To see more clearly the improvement the X Axis motion was captured in real-time (180us sample rate) and plotted. From the position the Velocity and Acceleration were also computed and plotted.

First the original captured motion X axis position, velocity, and acceleration vs. time through the original 3-degree facets with no corner rounding.

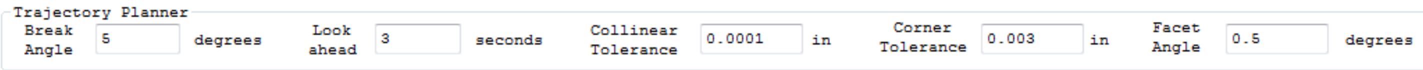
Note that the velocity plot has an odd shape for several reasons. At constant velocity the facet angle change causes the x velocity to drop while the y velocity increases, but since we are accelerating through the curve (because a more diagonal direction can make use of the increased (combined) acceleration 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.

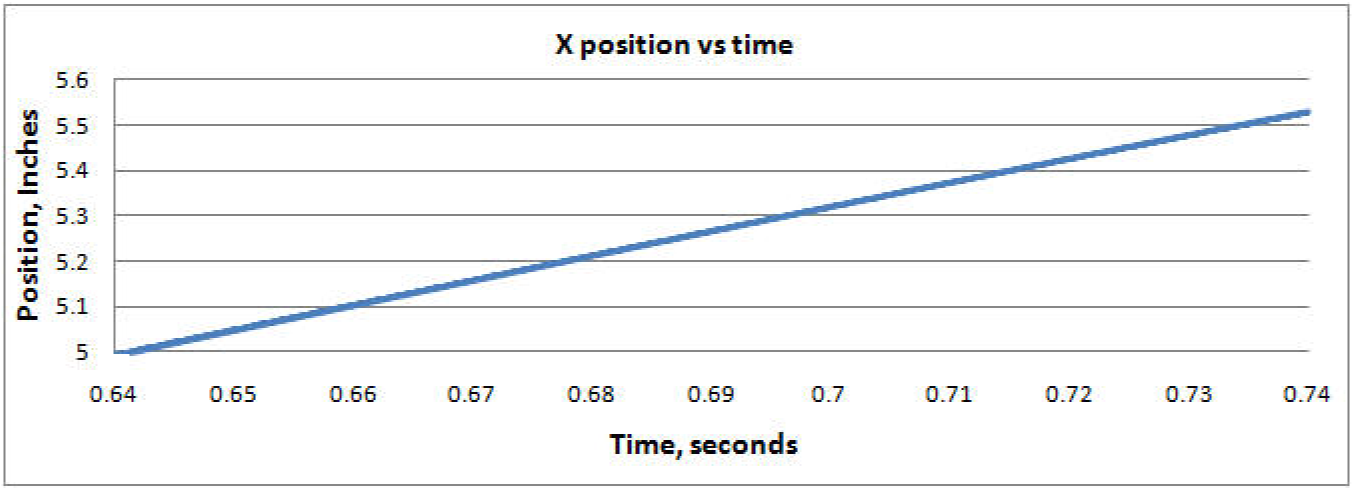


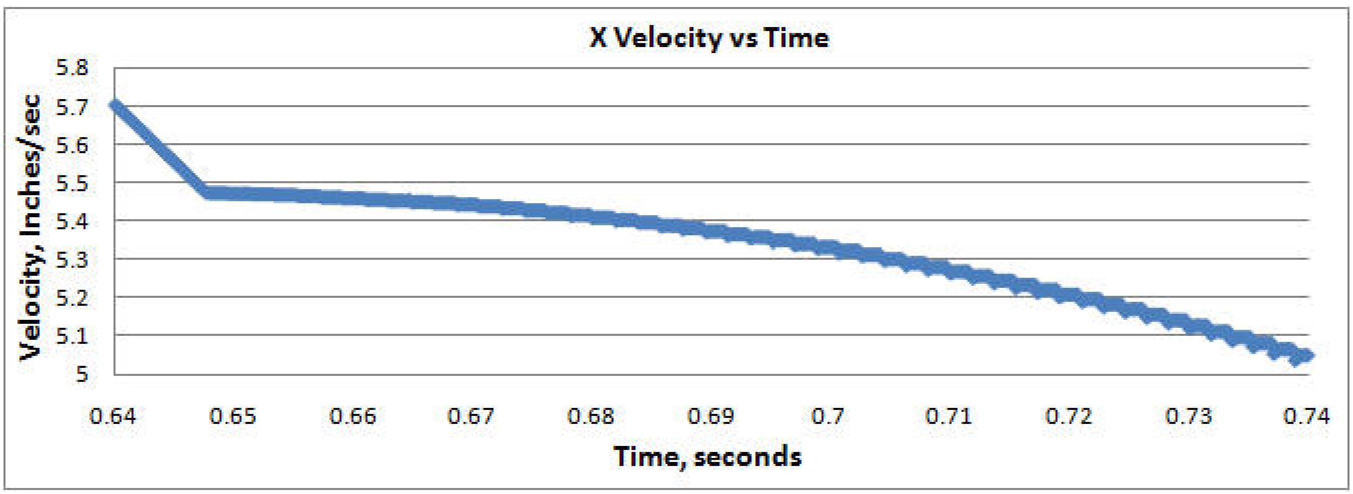


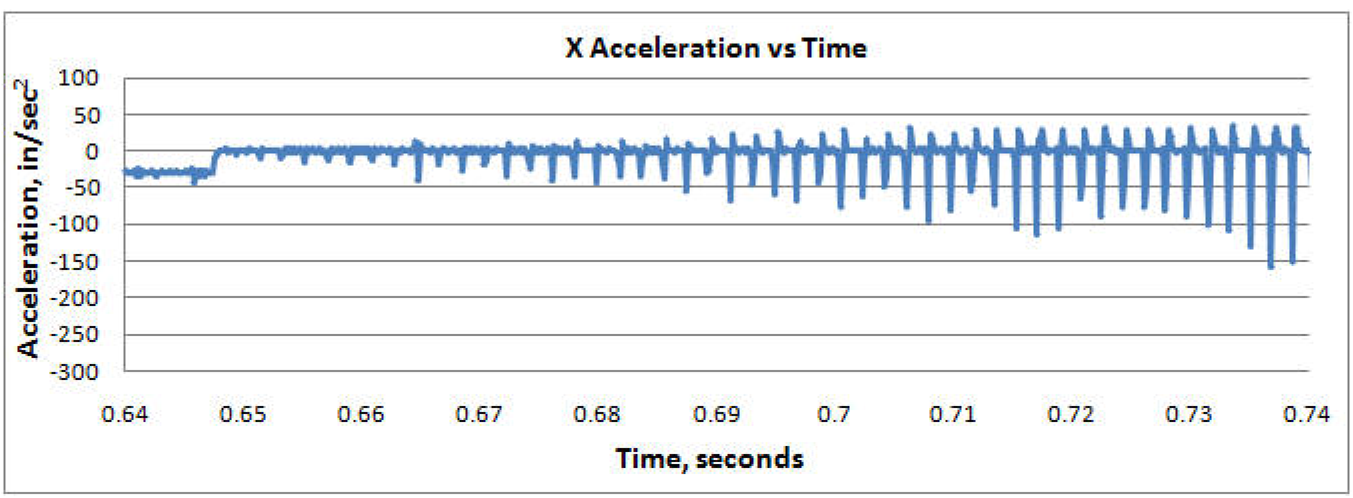


Now using the Trajectory Planner settings shown below. Notice the velocity is much smoother and the Acceleration is less Jerky.


Trajectory Planner
Break Angle
5
degrees
Look ahead
3
seconds
Collinear Tolerance
0.0001
in
Corner Tolerance
0.003
in
Facet Angle
0.5
degrees


X position vs time
Position, Inches
5.6
5.5
5.4
5.3
5.2
5.1
5
0.64
0.65
0.66
0.67
0.68
0.69
0.7
0.71
0.72
0.73
0.74
Time, seconds


X Velocity vs Time
Velocity, Inches/sec
5.8
5.7
5.6
5.5
5.4
5.3
5.2
5.1
5
0.64
0.65
0.66
0.67
0.68
0.69
0.7
0.71
0.72
0.73
0.74
Time, seconds


X Acceleration vs Time
Acceleration, in/sec^2
100
50
0
-50
-100
-150
-200
-250
-300
0.64
0.65
0.66
0.67
0.68
0.69
0.7
0.71
0.72
0.73
0.74
Time, seconds

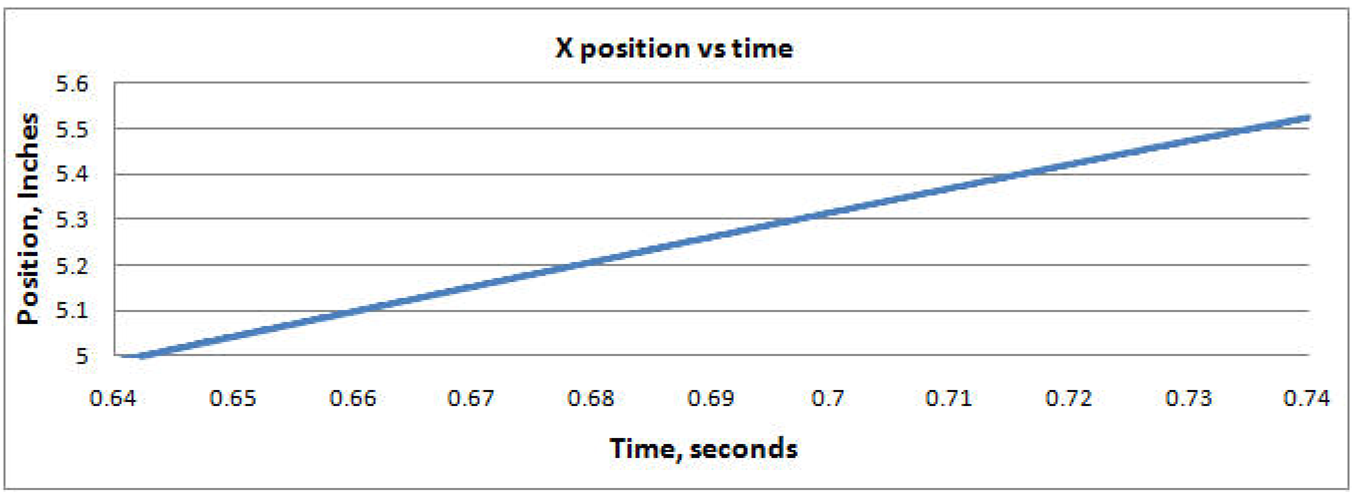
For still further motion smoothing an additional KFLOP feature can be used. A low pass filter can be applied to the output of the coordinated motion path.

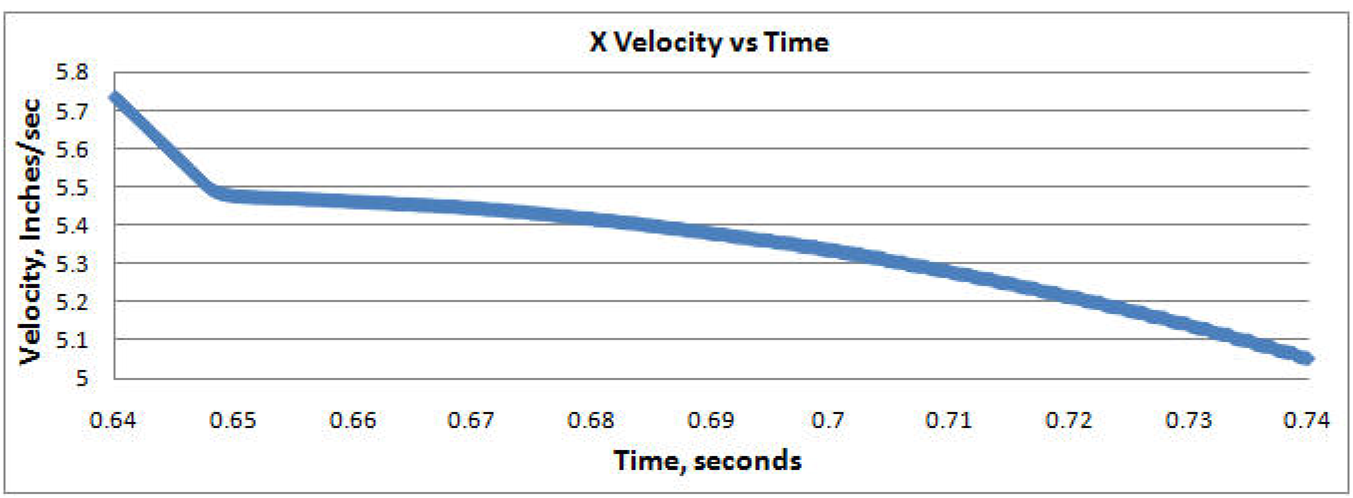
The Low Pass Filter will be applied to all axes of coordinated motion (up to 8) by setting the KLP coefficient within KFLOP. Currently a C Program must be used to set this global parameter.

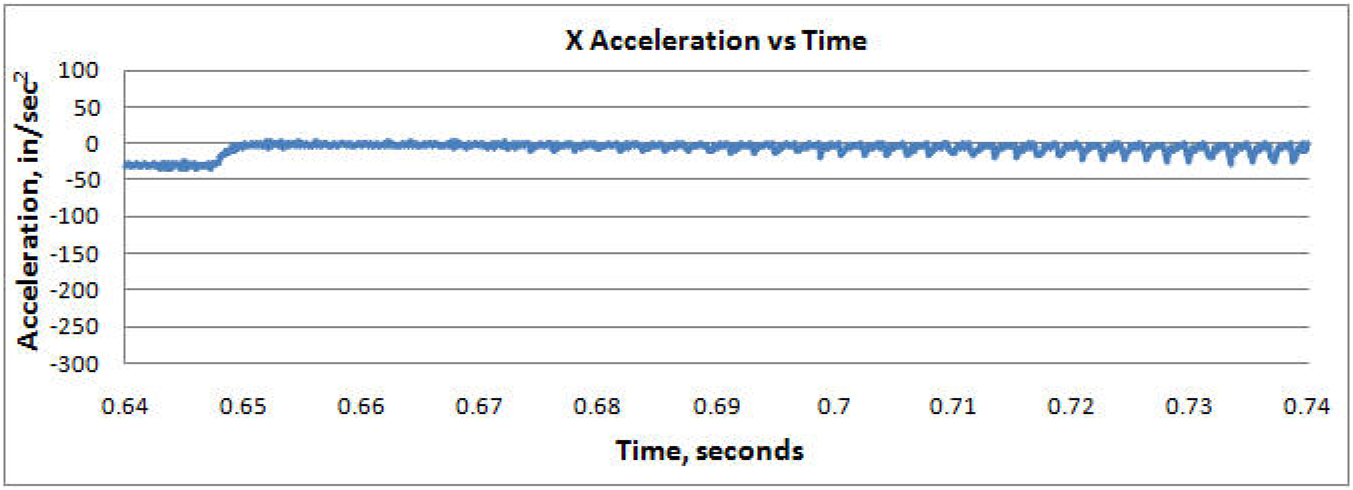
To compute an appropriate coefficient from a time constant Tau in seconds use the formula KLP = exp(-TIMEBASE/Tau) as shown below

```
#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.

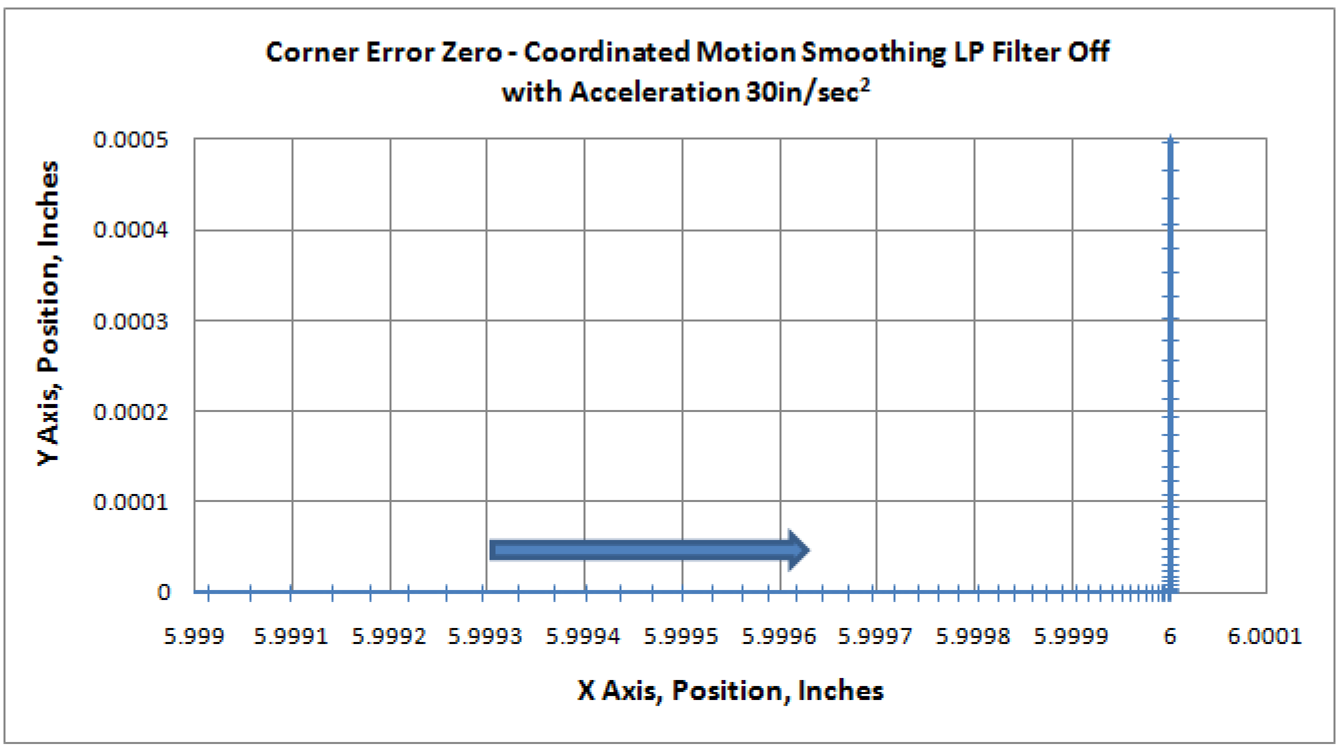




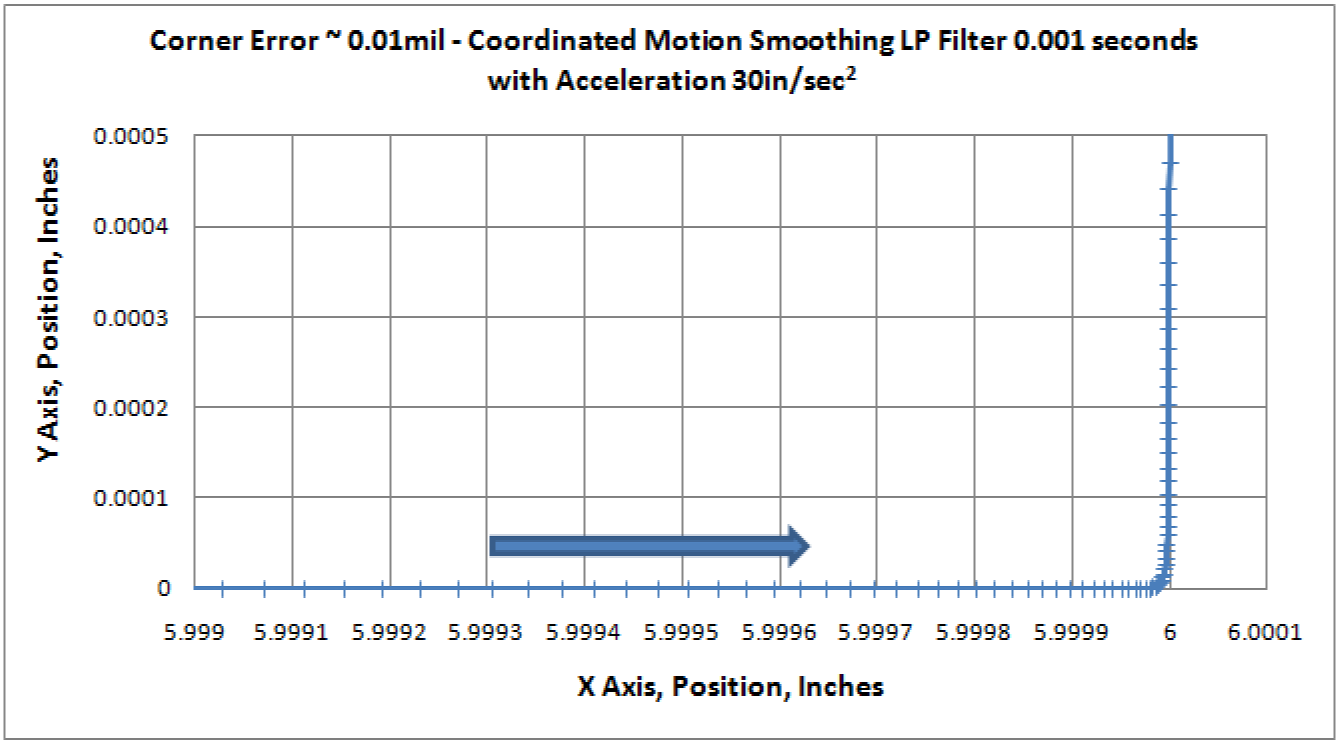


The disadvantage associated with low pass filtering of the trajectory is a potential lag in the commanded positions which may cause small path errors. The plots shown below show that a Low Pass Filter setting of 1 millisecond will be insignificant for most systems. A nearly worst case 90-degree angle with max deceleration on the X axis, followed by max acceleration on the Y axis are shown below.

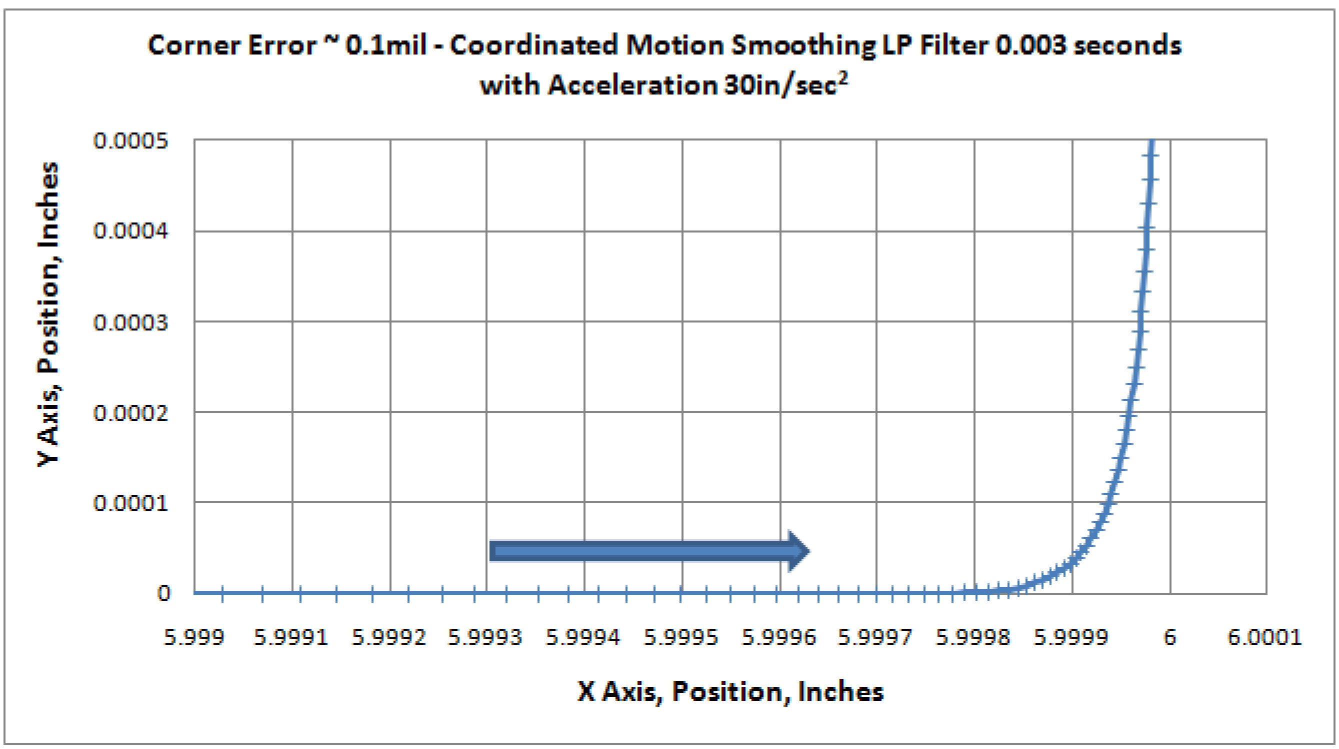
The first case is captured with no Low Pass Filtering. Note a perfectly square corner. Each tick mark shows a captured 180us sample point.



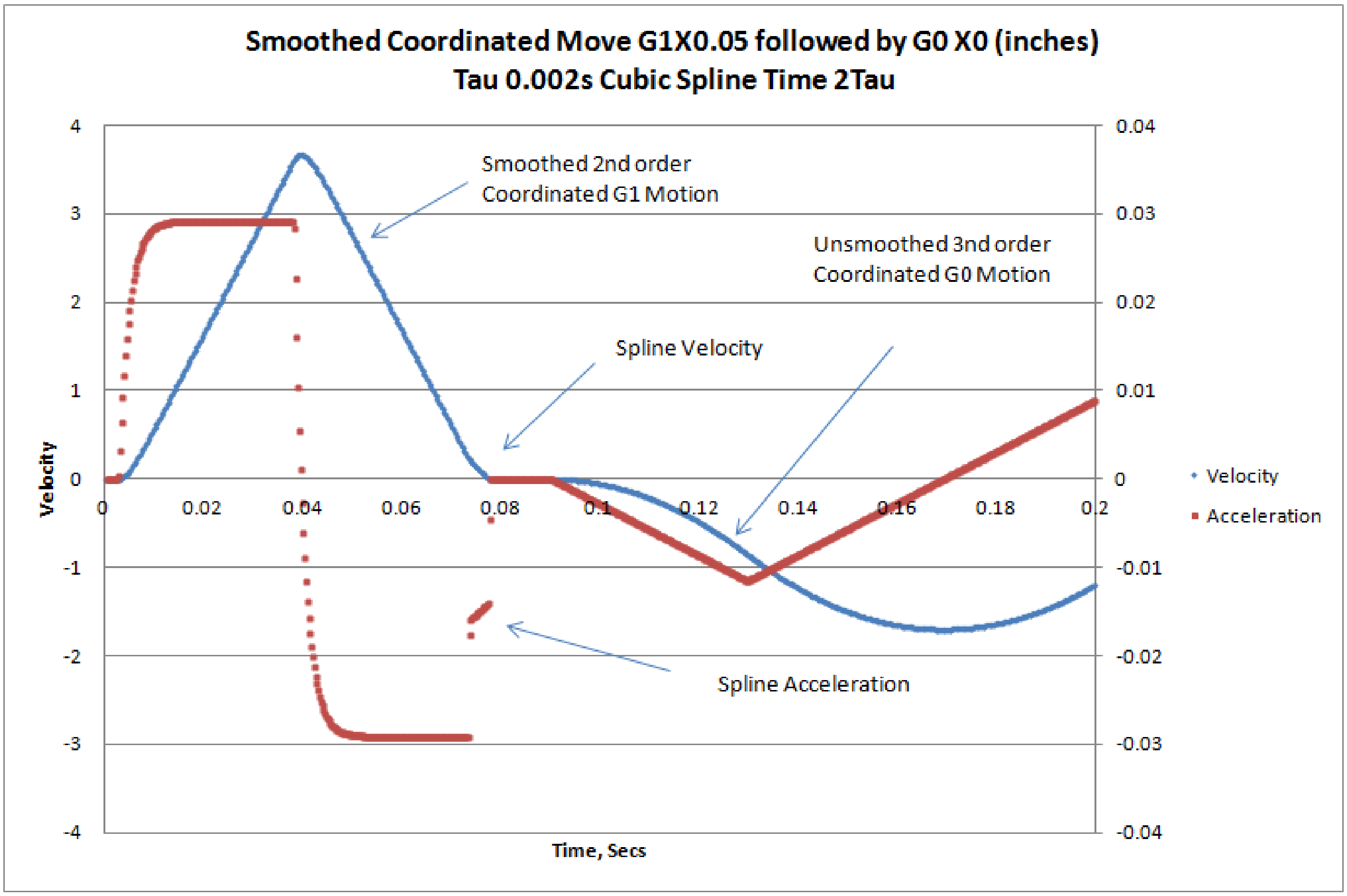
The case below is with 1 millisecond of filtering with a path error of ~ 0.01 mil (0.25um)



The case below is with 3 milliseconds of filtering with a path error of ~ 0.1 mil (2.5um)



Because the Low Pass Smoothing introduces a small position lag the very end of the smoothed path will not be completed when the coordinated path is completed. To complete the path the final position and velocity of each of the coordinated axes are used to calculate a Cubic Spline to complete the final small motion. The Cubic Spline allows continuous velocity from the end of the smoothed path to the target position in a manner where the velocity also reaches zero at the target position. A Time duration of the Cubic Spline set at 2xTau of the low pass filter provides a relatively constant deceleration to the Target.



Controlling KMotionCNC from KFLOP/Kogna

User KFLOP/Kogna C Programs can request actions to be performed by the KMotionCNC Application running on the PC by setting various command codes into special persist.UserData variables. The special UserData variables are being continuously uploaded with the Bulk Status record that KMotionCNC requests at approximately 10 times per second to update the DROs and so forth. If a command code is uploaded, KMotionCNC attempts to perform the requested action, then changes the UserData Command Value to a result code. Zero indicates success and a negative value indicates an error code (all commands are positive). So the typical process involves:

#1 - KFLOP/Kogna stores command into a Persist Var to request an action

#2 - The command is uploaded to KMotionCNC with the next status request

#3 - KMotionCNC performs the action

#4 - KMotionCNC clears the Persist Var to indicate completion

#5 - KFLOP/Kogna detects the command Var has been cleared to know the action was successful and complete

The Status uploads several Persist Vars which permits additional parameters to be uploaded with the command to KMotionCNC if required by the action to be performed. If extra parameters or data is required, then one of the uploaded parameters will specify where that data is located.

The number of UserData variables has now been expanded from 100 to 200 Variables and Variables 100-107 are the special vars constantly uploaded with bulk status. This is defined in the PC-DSP.h file as:

```
#define PC_COMM_PERSIST 100  // First Persist Variable that is uploaded in status
#define N_PC_COMM_PERSIST 8  // Number of Persist Variables that are uploaded in status
```

Currently supported actions include:

- EStop
- Halt
- Execute
- Single Step
- Set FRO
- Inc/dec FRO
- Set X,Y,Z,A,B,C DROs
- Push a User Defined Action Button
- Execute an M Code
- Display a Message Box
- Get/Set GCode #Vars

- Execute a MDI line of GCode

A new example called KFLOPtoPCCmdExamples.c is included which demonstrates how to invoke these actions from a KFLOP User C Program. The following helper functions are included in the example that simplify invoking the actions by setting the proper persist variables. They are:

```
// Trigger a message box on the PC to be displayed
// defines for MS Windows message box styles and Operator
// response IDs are defined in the KMotionDef.h file
int MsgBox(char *s, int Flags)

// put the MDI string (Manual Data Input - GCode) in the
// gather buffer and tell the App where it is
int MDI(char *s)

// Put a Float as a parameter and pass the command to the App
int DoPCFloat(int cmd, float f)

// Put an integer as a parameter and pass the command to the App
int DoPCInt(int cmd, int i)

// Pass a command to the PC and wait for it to handshake
// that it was received by either clearing the command
// or changing it to a negative error code
int DoPC(int cmd)
```

The Example code to make use of the helper functions is in the example as:

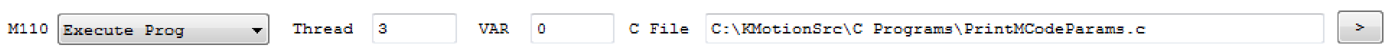
```
main()
{
	int Answer;
	double *pD = (double *)persist.UserData;

	DoPC(PC_COMM_ESTOP);
	DoPC(PC_COMM_HALT);
	DoPC(PC_COMM_EXECUTE);
	DoPC(PC_COMM_SINGLE_STEP);
	DoPCFloat(PC_COMM_SET_FRO,0.25f);
	DoPCFloat(PC_COMM_SET_FRO_INC,1.1f);
	DoPCFloat(PC_COMM_SET_X,0.0);
	DoPCFloat(PC_COMM_SET_Y,0.0);
	DoPCFloat(PC_COMM_SET_Z,1.25);
	DoPCInt(PC_COMM_USER_BUTTON,3);
	DoPCInt(PC_COMM_MCODE,3);

	Answer = MsgBox("Hello World",MB_YESNO|MB_ICONEXCLAMATION);
	if (Answer == IDYES)
		printf("Answer is Yes\n");
```

```
	else
		printf("Answer is No\n");


	MDI("G0 X1.2 Y2.2 Z3.3");


	// put 3 double values in the persist vars

	pD[10] = 123.456;
	pD[11] = 1000.0;
	pD[12] = 999.9;

	// transfer up to the GCode Vars
	SetVars(100,3,10);  // Upload 3 to GCode 100 from persist 10

	MDI("#100 = [#100 + 1]");

	// read them back into different persist Vars
	GetVars(100,3,13);  // Download 3 from GCode 100 to persist 13

	printf("%f %f %f\n",pD[13],pD[14],pD[15]);
}
```

MCodes with Parameters

KMotionCNC allows MCodes 100-119 with parameters when configured to Execute a KFLOP User C Program. The P Q and R words in the GCode block will be downloaded to KFLOP's persist.UserData variables before the C program is executed. The parameters will be placed into consecutive KFLOP variables as 32-bit floating point values starting with the variable specified in the MCode Configuration. Zero, one, two, or all three parameters may be specified in the GCode block (line). The order they are placed into the variables will always be in the P Q R order where any or all of the parameters will be omitted if they are not explicitly specified in the GCode Block. If no parameters are specified then the MCode number will be stored into the one Var specified.

Example M Code Configuration



Example M Code Usage with all 3 parameters

M110 P1.23 Q4.56 R-1

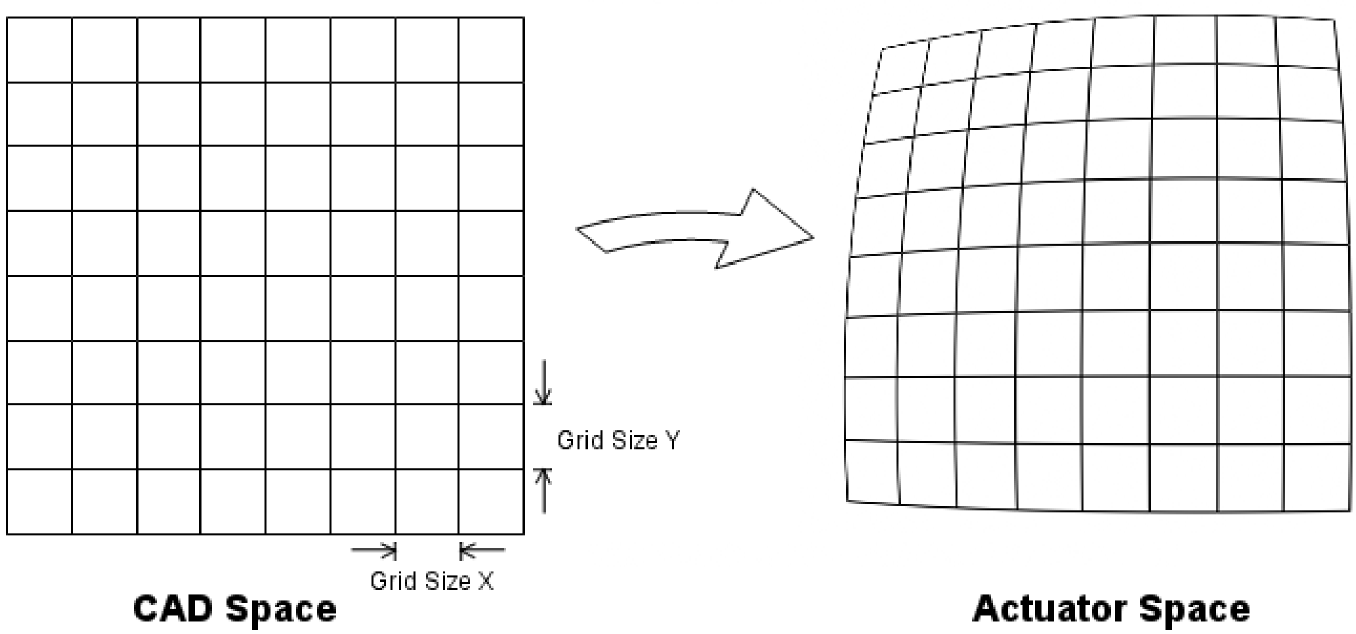
Example C Code accessing parameters:

```
#include "KMotionDef.h"

main()
{
	printf("P = %f Q = %f R = %f\n",
		*(float *)&persist.UserData[0],
		*(float *)&persist.UserData[1],
		*(float *)&persist.UserData[2]);
}
```

Printed Result

P = 1.230000 Q = 4.560000 R = -1.000000

Geo Correction Table



The area of the CAD Space to be corrected is defined by a number of rows and columns and an XY Grid size. A CAD point falling within a grid uses a bilinear interpolation from the 4 points making up the grid region to map to actuator space.

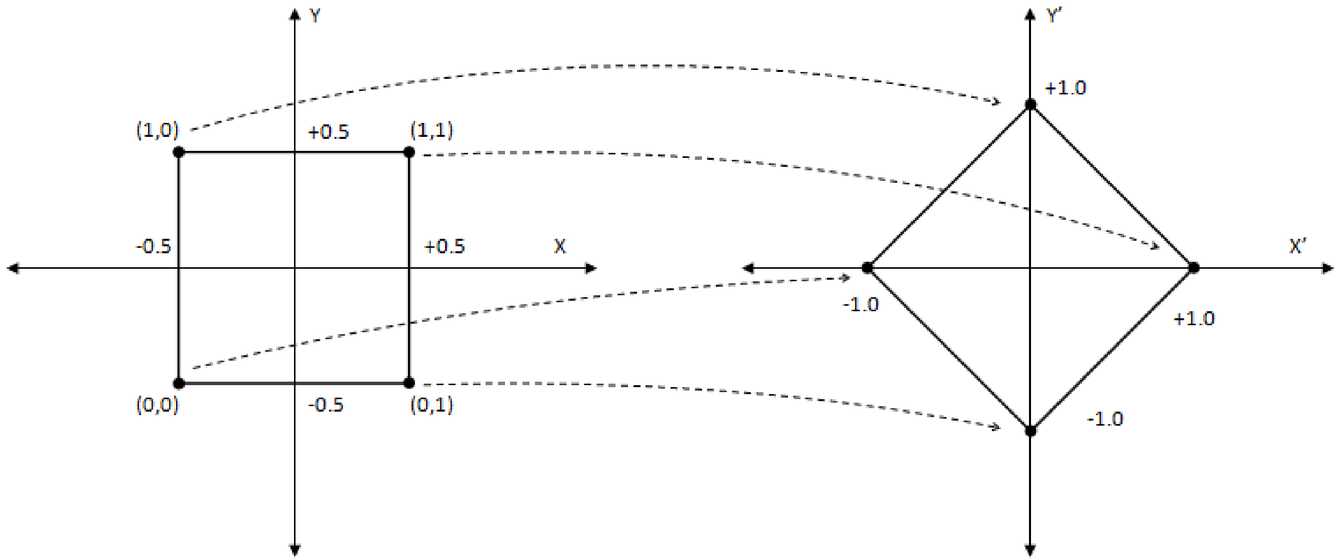
There is also a Z coordinate associated with each grid point. The mapping applies this as a Z offset in order to flatten the XY plane if necessary. If no flattening is required then the z values should all be specified as zero in the table.

Normally a Geo Correction Table is defined to be large enough to map the entire range of motion of the system. However, for points outside the range of the Table the closest grid region is extrapolated to form the result. The largest Table currently allowed is a 4000x4000 array.

When a Geo Correction Table has been loaded Arcs are automatically broken into small line segments internally where each segment's endpoints are Geocorrected into actuator space. This is necessary because an Arc in CAD space in unlikely to still be an Arc in Actuator Space. The Trajectory Planner Collinear Tolerance Setting is used to determine how small of line segments are used. Line segment lengths will be chosen small enough to not deviate from the true Arc by more than the Collinear Tolerance. For most systems a value of 0.001 inches should be used.

The smallest possible Geo Table consists of a single rectangle consisting of 4 points (2 rows and 2 columns). In this case the entire infinite CAD Space plane is mapped to the Actuator Space using a single set of transformation equations.

The Example below shows how the table can be used to form a simple transformation:



Theoretical GCode CAD Space *Transformed Geo Corrected Space*

Example Transformation

X' = X + Y

Y' = X - Y

row	col	Y	X	X' = X + Y	Y' = X - Y
0	0	-0.5	-0.5	-1	0
0	1	-0.5	0.5	0	1
1	0	0.5	-0.5	0	-1
1	0	0.5	0.5	1	0

Corresponding Geo File

Simple 2x2 grid on CAD Space Grid Spacing of 1 inch in X and 1 inch in Y.

Then 4 grid points (row,col,X,Y,Z)

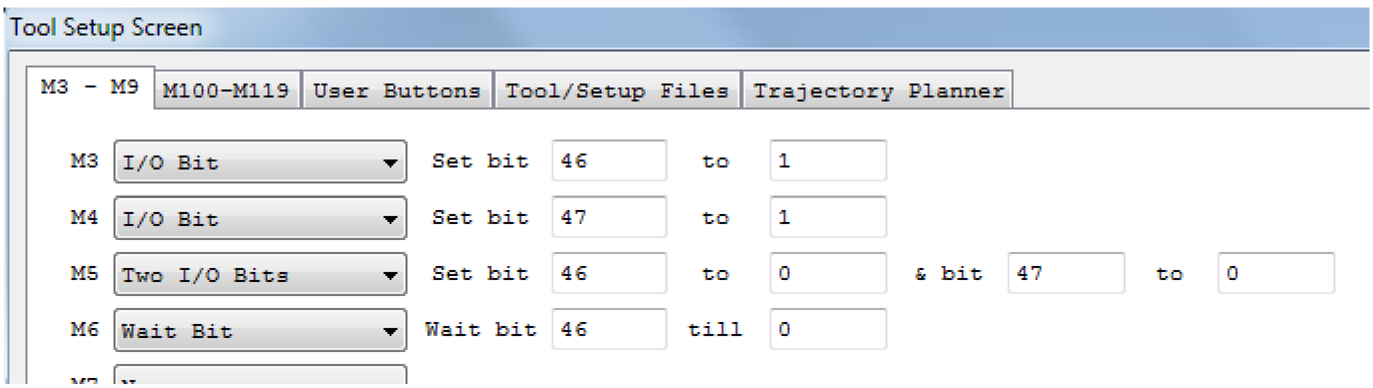
See file as <Install>\KMotion\Data\MeasurementsPlusMinus.txt

```
2,2
1,1
-0.5,-0.5
0,0,-1, 0, 0
0,1, 0,-1, 0
1,0, 0, 1, 0
1,1, 1, 0, 0
```

Synchronous IO Commands Embedded in Coordinated Motion

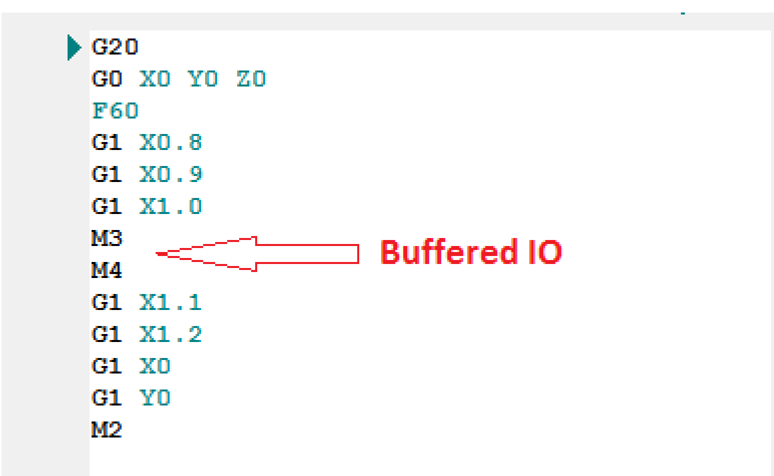
KMotionCNC allows I/O operations to be embedded into the Coordinated Motion Buffer such that the IO commands are output synchronously (within 90us Servo Sample) with motion. (Buffered IO operations can also be inserted with Library calls from custom applications).

This example shows two MCodes configured to set bits I/O bits 46 and 47 high and also MCode M6 configured to wait for I/O bit 46 to be low.



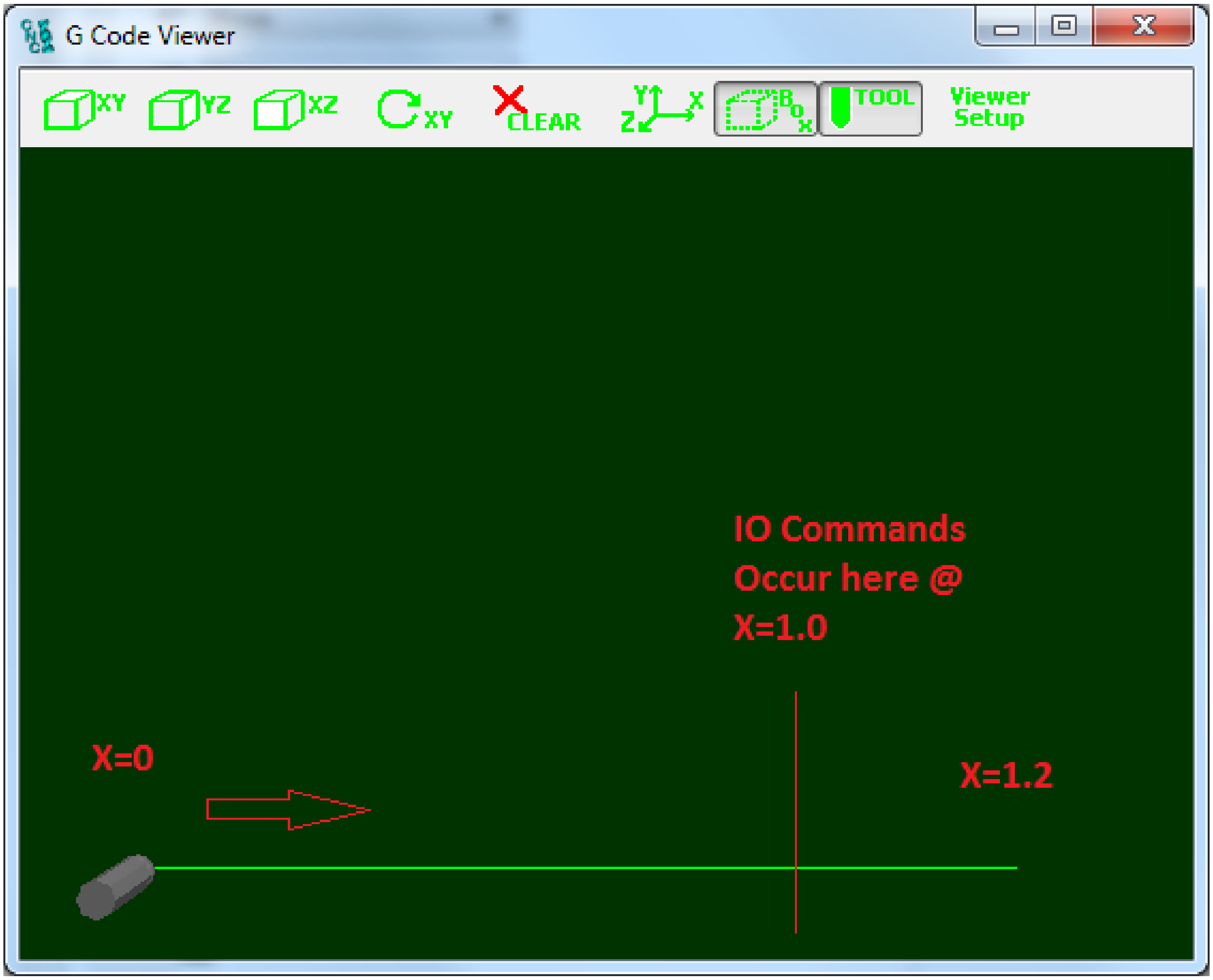
A simple GCode Fragment shows the MCodes embedded within a continuous straight motion right at the point the motion reaches X=1.0.

The Trajectory Planner normally combines very collinear motion blocks together for smoothing and so forth. But in this case it is careful to not combine segments that cross a buffered I/O boundary.



Executing the GCode we can observe the IO bits 46 and 67 set high as the motion crosses X=1.0

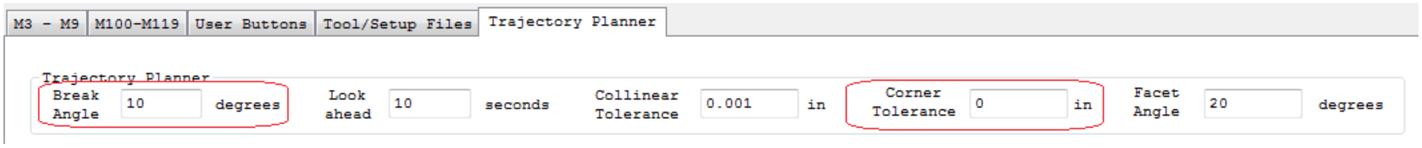
Executing the GCode we can observe the IO bits 46 and 67 set high as the motion crosses X=1.0



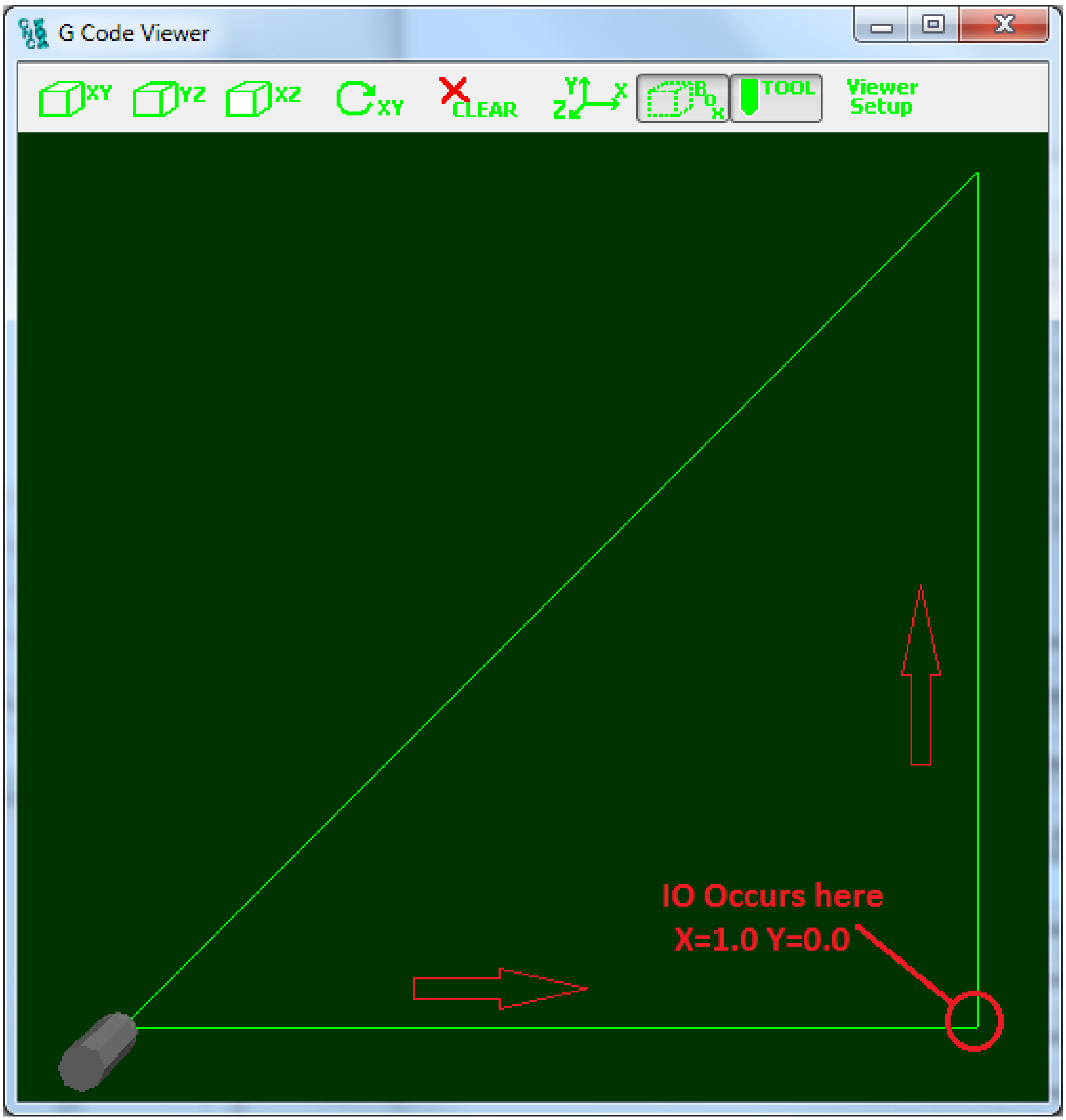
The GCode below demonstrates a right-triangle motion path with the I/O commands inserted at a corner.

```
G20
G0 X0 Y0 Z0
F60
G1 X0.8
G1 X0.9
G1 X1.0
M3
M4
G1 Y0.1
G1 Y0.5
G1 Y1.0
G1 X0Y0
M2
```

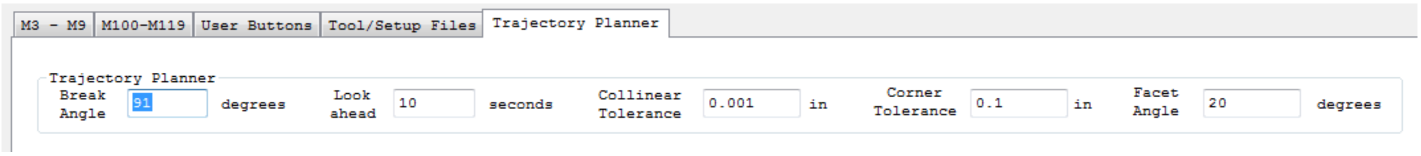
With Trajectory Planner Break Angle set to 10 degrees (which is less than 90 degrees) a full stop will occur at the corners.



In this case the IO switches where the instantaneous stop occurs at a corner.



To demonstrate what happens with corner rounding an exaggerated corner rounding example is shown below. With a Break Angle greater than 90 degrees a stop will not occur at the corner. Also a large radius (0.1 inches) and large facet angle (20 degrees) corner rounding is configured.

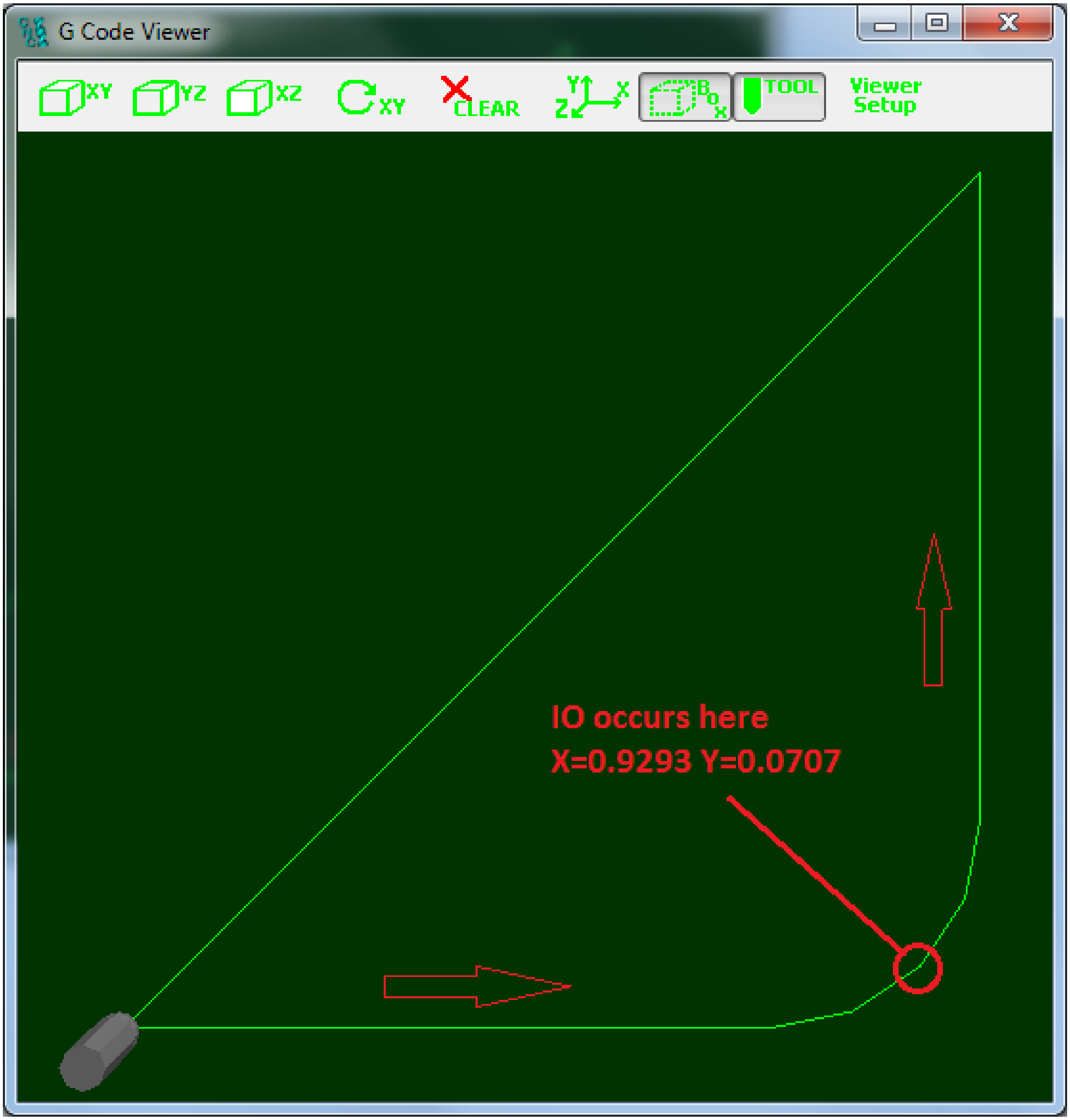


To observe exactly what happens we use the KFLOP C Program shown below to real-time capture the XY position when the IO occurs (XY resolution is 10000 counts per inch).

```
#include "KMotionDef.h"

main()
{
	int New,Last=ReadBit(46);
	for (;;)
	{
		New = ReadBit(46);
		if (New != Last)
		{
			Last=New;
			printf("X=%f Y=%f\n",ch0->Dest/10000.0,ch1-
>Dest/10000.0);   // send message to console
		}
	}
}
```

The captured position is printed as:

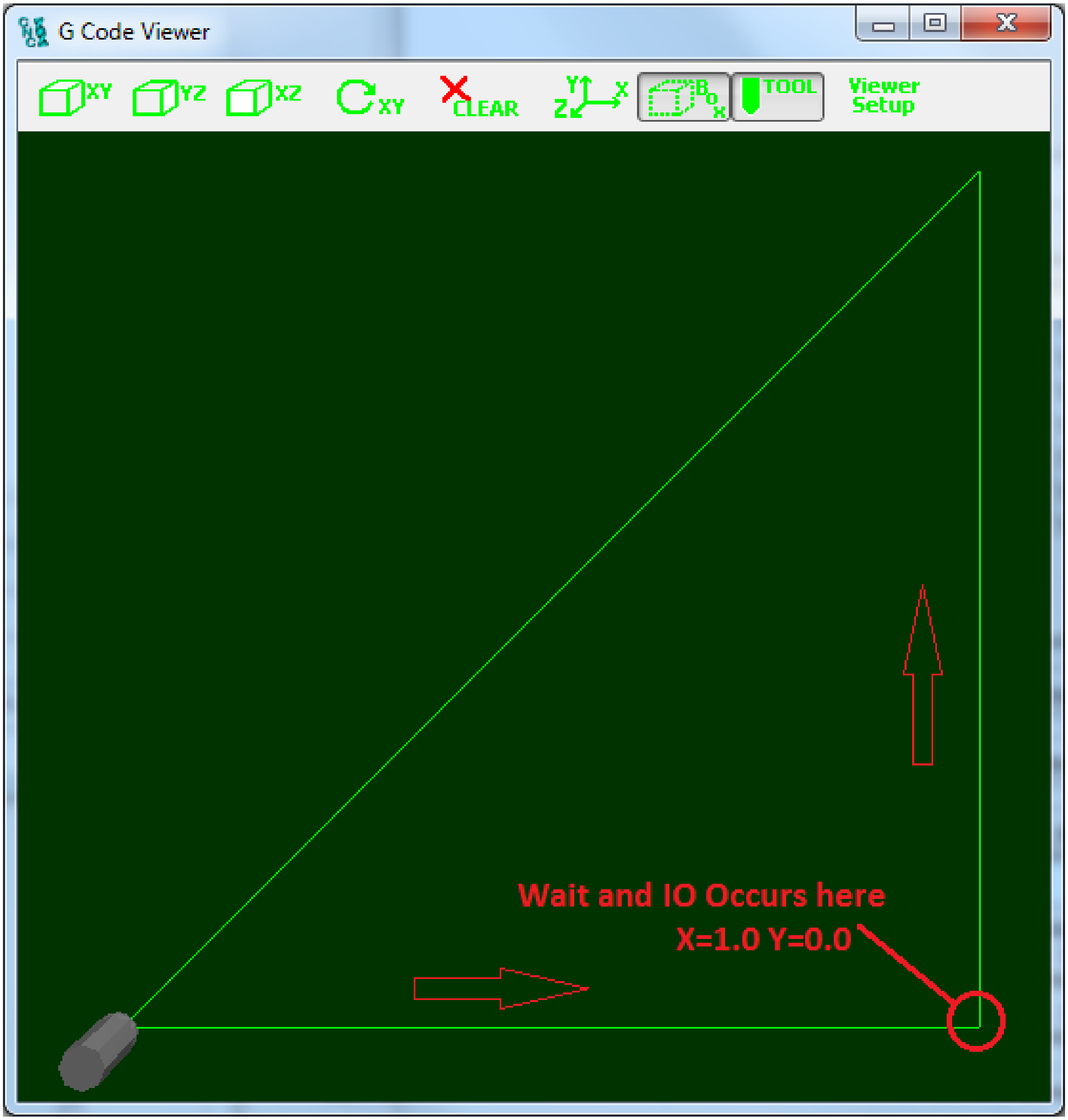
X=0.929315 Y=0.070748



In addition to embedding outputs into the motion stream, waits for inputs can be embedded. This is inserted as a buffered WaitBitBuf command. When executed if the specified bit is false the motion will stall at that point until the input becomes true. There is a similar command WaitNotBitBuf that will stall until the input becomes false. MCodes can be configure to insert these commands, Although Wait commands can be inserted anywhere in a motion path it normally is only useful to place them at locations where the motion stops. Such as the very beginning of a path or at a corner where motion comes to a stop. Otherwise an instantaneous stop will occur without any acceleration. Wait commands are useful when motion must proceed instantly on command. This is possible because the motion has already been Interpreted, planned, downloaded, and commanded to execute ahead of time. See the example GCode below where a wait has been inserted at a corner.

```
G20
G0 X0 Y0 Z0
F60
G1 X0.8
G1 X0.9
G1 X1.0
M3
M6 (wait for bit 46 low)
M4
G1 Y0.1
G1 Y0.5
G1 Y1.0
G1 X0Y0
M2
```

The plot below shows where the wait will stall execution if the specified bit is false.


G Code Viewer
XY
YZ
XZ
XY
CLEAR
TOOL
Viewer
Setup
Wait and IO Occurs here
X=1.0 Y=0.0

KMotionCNC Spindle Control

Overview

KMotionCNC allows the User to configure how the Spindle is to be be controlled. Four basic actions that need to be defined are:

M3 - Turns on the Spindle in a CW direction

M4 - Turns on the Spindle in a CCW direction

M5 - Turns off the Spindle

Snnnn - Sets the desired RPM or Constant Surface Speed in feet/min or meters/min

To configure what action is to take place for each of the four functions the Tool Setup must be configured to either do the operations directly, or by invoking a KFLOP User C Program.

For the most basic cases where only one or two control IO bits are required for On/Off/Direction the IO operations can be configured directly. If Speed can be controlled by simple open loop scaling/limiting of a DAC output then speed control can also be configured directly.

For more complex cases and to allow control of virtually any type of Spindle, through any type of interface, KMotionCNC can be configured to invoke User programs to perform the four tasks listed above. Spindles might need to be controlled in a variety of different ways: Open Loop, Closed Loop, DACs, PWMs, Step/Dir, ModBus, Rs232, etc... Although this requires a small program to be written, it provides maximal flexibility and full control to the system designer. Usually with only a few lines of code the desired task may be performed. Common examples are provided.

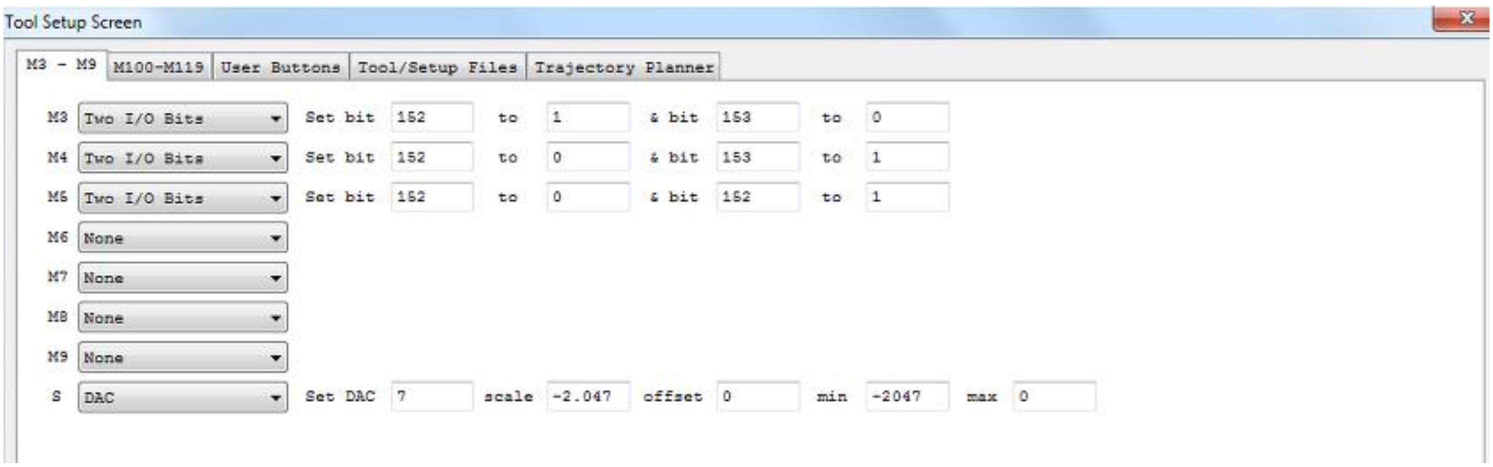
For G96/G97 - CSS (Constant Surface Speed) an additional program is required to run continuously in KFLOP which changes spindle speed continuously as a function of X position (radius). See the CSSJog.c example.

If a quadrature encoder is installed on the spindle it may be used for Spindle Speed Measurement and Display. See the Threading Configuration described here.

Basic Configuration

For basic configurations where the spindle CW/CCW/OFF can be controlled with one or two IO bits (normally Kanalog Relay Driver or Opto Outputs) and with speed control from a Kanalog DAC then no C programming is required. An example configuration is shown below. Note the S action is mapped to Kanalog DAC #7 with a negative scale of -2.047. Kanalog DAC outputs are inverting such that negative DAC counts result in a positive voltage. The assumption in this case is that an S value of 1000RPM would correspond to -2047 DAC counts or +10V. And the allowed range is from -2047 to 0 counts (0V to +10V).

An additional advantage to using I/O bits rather than C Programs is that they I/O commands will be embedded within the motion stream and will occur synchronously with the motion and without any pause in the motion. This is especially useful for faster systems with Lasers/Engravers/Extruders instead of a traditional Spindle. For more information on buffered IO commands see here.

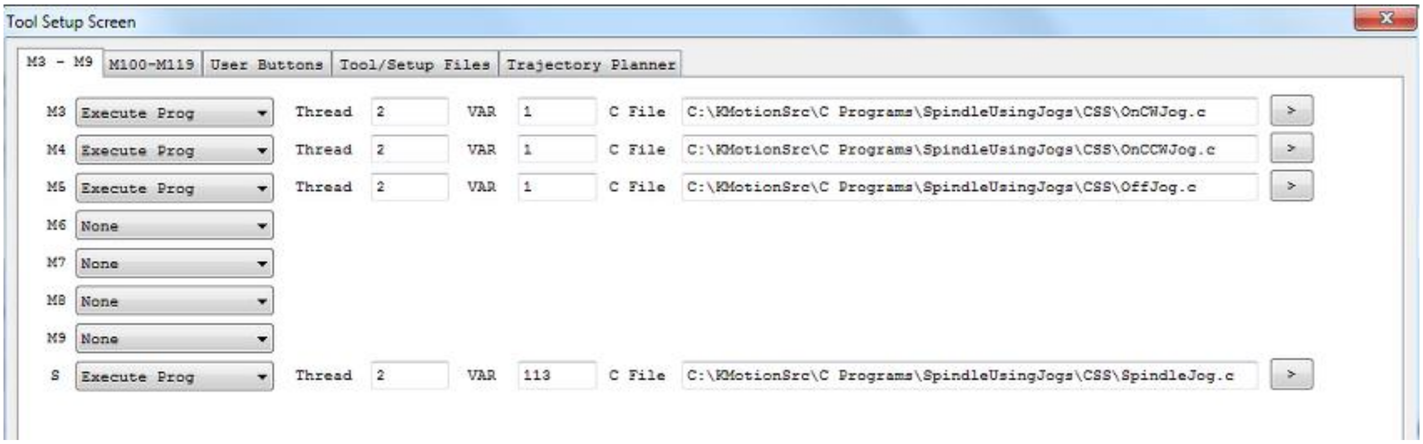


User Program Configuration

For more advanced functionality User C Programs may be assigned for each of the spindle actions. For systems that have a spindle that can be controlled with an axis channel the C Programs listed below can be used directly with little or no modification. Note if your Spindle speed is controlled by a DAC an Axis channel can be configured to output open loop to a DAC, see here for how. This assumes that axis channel has already been configured much like the other axes in the system and can be moved using Axis Jog Commands. Spindles such as this are types that can be driven like a servo with feedback, or as an open loop Step/Dir drive. The only requirement is that a KFLOP axis channel can be used for control. This also allows acceleration (and Jerk) to be controlled by the axis channel for smooth speed changes.

The example configuration below shows the four spindle tasks assigned to four C programs. Each of the programs run temporarily to perform their function and should never execute concurrently so each can use the same Thread. See here for more information on Threads. M3, M4, and M5 do not pass any parameter so the VAR should be assigned to ay unused Variable such as #1. For S the speed value is passed in the VAR parameter. If G96/G97 CSS is to be used then the same VAR should be used as the G96 speed is passed. This VAR number is defined in PC-DSP.h as

```
#define PC_COMM_CSS_S 113 // S speed setting in inches/sec
```



In most cases the four C programs can be used without any modification. The programs all include a common file called MySpindleDefs.h that contain settings that are likely to require customization for any particular system. The file and parameters are defined and described below. Each of the four C programs includes this file with an include statement of: #include "MySpindleDefs.h". This should be placed immediately after the #include "KMotionDef.h" (which includes and defines all the standard KFLOP definitions).

Some Spindle controls/interfaces can accept a positive and negative command to drive in CW and CCW directions. Others require inputs (relays) to switch the directions and the speed is always set as a positive command. If your system is capable of accepting positive and negative commands then define

USE_POS_NEG_VOLTAGE as 1 otherwise define it as 0.

MySpindleDefs.h

```
#define SPINDLEAXIS 6 // Axis Channel to Jog to rotate Spindle
#define FACTOR (1000/60.0) // to convert RPM to counts/sec (counts/rev / 60.0sec)
#define SPINDLECW_BIT 154 // bit to activate to cause CW rotation
#define SPINDLECCW_BIT 155 // bit to activate to cause CCW rotation
#define SPEEDVAR 99 // global persistant variable to store latest speed
#define STATEVAR 98 // global persistant variable to store latest state (-
1=CCW,0=off,1=CW)
#define KMVAR PC_COMM_CSS_S // variable KMotionCNC will pass speed parameter (113)
#define USE_POS_NEG_VOLTAGE 0 // 0 = output Magnitude, 1 = output positive and negative
speed
```

These C Programs (located in the <Install Directory>\C Programs\SpindleUsingJogs\CSS directory) are written in a manner to handle a common issue with Spindles which is that a Speed change while the Spindle is commanded off should not be immediately applied. The four programs use two global variables to maintain the latest state and speed. This allows a speed change (when off) to be delayed until the spindle is turned on, but also be applied immediately if already on. In the example above VARs 99 and 98 are used for this purpose.

CSS – Constant Surface Speed

Constant Surface Speed allows a lathe cutting tool to maintain a constant linear rate regardless of the current radius (or diameter) as the spindle RPM is continuously adjusted as a function of the radius.

Here is a simple GCode Program that uses G96 CSS mode

```
G90 G21              (Absolute mm)
G00 X50.8 Y0 Z50.8   (Rapid to starting position)
G96 M3 D2500 S159.6 (CSS should be 1000RPM at R=25.4mm)
G1 X2.54 Z25.4 F350 (Change X - radius)
G1 X50.8 Z0
G1 X2.54 F5000       (Change X - radius faster)
G1 X50.8 Z0
G1 X2.54             (Change X - radius faster)
G1 X50.8 Z0
M05                  (Spindle off)
G0 Z50.8
G97 M3 S500          (Run at normal RPM Mode)
G4 P3
M2
```

G96/G97 CSS requires some additional C Program functionality to be running within KFLOP. When KMotionCNC decodes a G96 command to enter CSS mode it does so by setting a number of VAR parameters within KFLOP. A program continuously running within KFLOP will detect the CSS mode and set the Spindle RPM as a function of the X axis position (radius) and requested surface speed. The five variables set by KMotionCNC to control CSS are listed in the shared header file PC-DSP.h as and define which persist variables the data will be placed:

```
// Persist Variable used with CSS Constant Surface speed for Spindle
// These parameters will be set when GCode switches to G97 CSS mode
// A User program in KFLOP must be running to monitor the mode an x position
// and alter the spindle RPM appropriately.
#define PC_COMM_CSS_MODE 110 // Mode 1=Normal RPM mode. 2=CSS
#define PC_COMM_CSS_X_OFFSET 111 // X axis counts for Radius zero
#define PC_COMM_CSS_X_FACTOR 112 // X axis factor to convert counts to inches
#define PC_COMM_CSS_S 113 // S speed setting in inches/sec
#define PC_COMM_CSS_MAX_RPM 114 // Limit max RPM to this value as Radius approaches
zero
```

MODE: defines whether CSS is active or not. G96 sets all CSS parameters and then switches the mode to 2. G97 (normal fixed RPM Mode) sets the Mode to 1. On Power up the Mode will be 0. X_OFFSET allows KFLOP to relate a position in counts to a radius from the spindle axis. It consists of all GCode Offsets converted to X axis counts. X_FACTOR is the inverse of the X axis resolution which allows KFLOP to do a simple multiplication to convert X axis counts to inches. S: is the desired surface speed in inches/sec (converted from GCode units of feet/min or meters/min). MAX_RPM: specifies the maximum allowed RPM. As the radius approaches zero the required RPM to maintain a specified surface speed will approach infinity. Any computed RPM beyond this value will be limited to this value. The max RPM may be specified on the line of GCode with the D word. If no D word is specified on the same line as the G96 then the limit will be sent as 1e9 RPM.

In most cases the supplied function ServiceCSS() shown below can be used. See C Program comments below on how to include this file into your Initialization C file and add a call to it in a continuous loop. If you already have a continuous loop in your Initialization C File then add a call ServiceCSS() within it. If there is not already an loop add one as shown in the comments below.

```
// Handle CSS (Constant Surface Speed) messages from KMotionCNC)
//
// This code assumes you have an Axis Channel Configured to control
// Spindle speed and Jog Calls can be made to control speed
//
// Include this function into your main Init Thead code and call it
// continuously from a forever loop similar to that shown here

//#include "KMotionDef.h"
//#include "MySpindleDefs.h"
//#include "CSSJog.c"
//main()
//{
//      for (;;)
//      {
//              WaitNextTimeSlice();
//              ServiceCSS();
//      }
//}

int   *css_mode = &persist.UserData[PC_COMM_CSS_MODE];              // Mode
1=Normal RPM mode. 2=CSS
float *css_xoff = &persist.UserData[PC_COMM_CSS_X_OFFSET];          // X axis
counts for Radius zero
float *css_xfactor = &persist.UserData[PC_COMM_CSS_X_FACTOR]; // X axis factor
to convert counts to inches
float *css_s = &persist.UserData[PC_COMM_CSS_S];                          // S
speed setting in inches/sec
float *css_max_rpm = &persist.UserData[PC_COMM_CSS_MAX_RPM];  // Limit max RPM
to this value as Radius approaches zero

double css_T=0;  // update only every so often
#define CSS_UPDATE_DT 0.05
```

```
void ServiceCSS(void)
{
    float rpm;
    double T=Time_sec();

    if (*css_mode == 2 && T > css_T) // check if we are in CSS mode and it is
time to update
    {
        css_T=T+CSS_UPDATE_DT;  // determine next time to update

        // convert axis position to distance from center in inches
        float radius = fast_fabs((chan[CS0_axis_x].Dest - *css_xoff) *
*css_xfactor);

        if (radius > 0.0f)
            rpm = *css_s / (radius * (TWO_PI_F/60.0f));
        else
            rpm = *css_max_rpm;

        if (rpm > *css_max_rpm) rpm = *css_max_rpm;

        if (persist.UserData[STATEVAR]!=0)  // if spindle is already on, ramp to
new speed
        {
            if (USE_POS_NEG_VOLTAGE)
               Jog(SPINDLEAXIS,rpm * FACTOR * persist.UserData[STATEVAR]);
            else
               Jog(SPINDLEAXIS,rpm * FACTOR);
        }
//      printf("xoff=%f radius= %f xfactor=%f s=%f(ips) maxrpm=%f
rpm=%f\n",*css_xoff,radius,*css_xfactor,*css_s,*css_max_rpm,rpm);
    }
}
```

Step-by-step Recap

This sequence is applicable for Spindles that can be driven with a KFLOP Axis Channel:

#1 - Electrically Interface your spindle

#2 - Using KMotion.exe configure your Spindle Axis Channel and test that KMotion Console Screen Jog commands can successfully control it.

#3 - Modify MySpindleDefs.h with settings that apply for your system

#4 - Configure KMotionCNC | Tool Setup | M3 - M9 | M3, M4, M5, and S

#5 - If Spindle encoder feedback is available configure KMotionCNC | Tool Setup | Trajectory Planner | Threading

#6 - if CSS is desired include and add ServiceCSS() to a continuous loop in your Initialization C File

Video

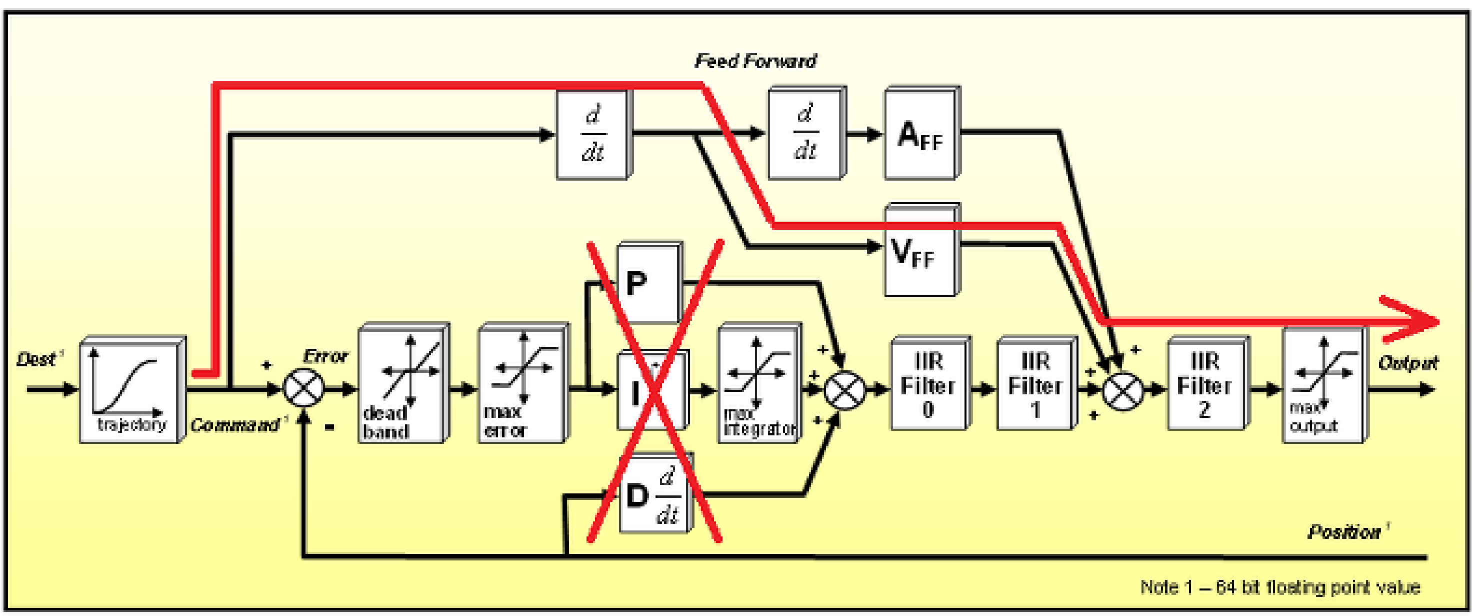
Simple demo Video of final GCode running CSS GCode.

User Video using CSS.

Configuring an Axis to Control a DAC open Loop

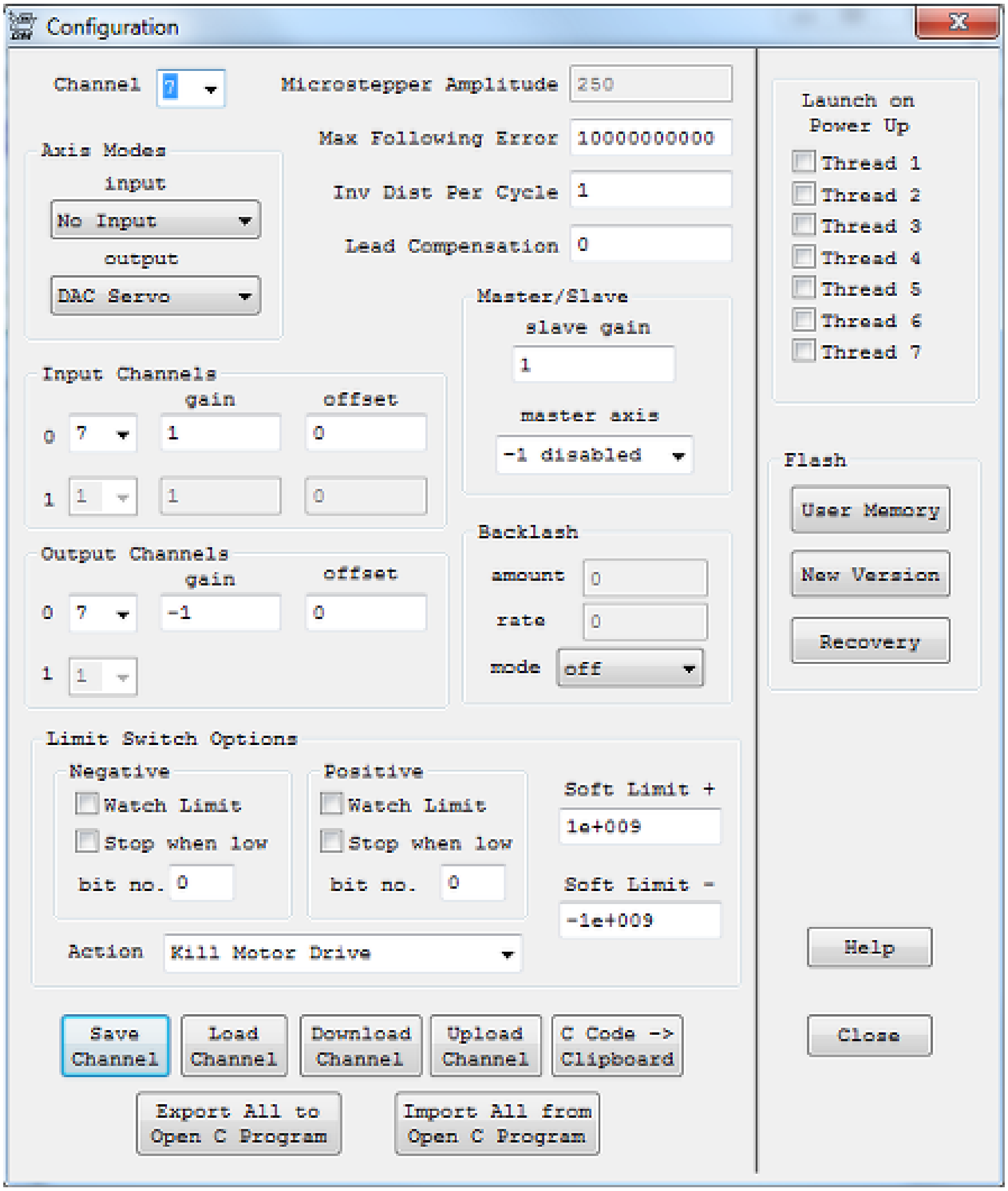
For Spindles that are controlled open loop with a 0-10V signal it is possible to configure an Axis Channel to output current Speed directly to a DAC. There a several advantages to configuring in this manner. First, the Spindle can be controlled using Jog Speed Commands in exactly the same manner as a closed loop servo (or any other) type of Axis Channel. And second the Axis Motion Profile Parameters (Acceleration and Jerk) can be used to ramp the speed up and down.

The configuration is the same as a DAC Servo with no Feedback (PID all zero) and with Velocity Feed Forward. The Servo diagram shows how the Motion Profile can be routed directly o the output using the Velocity Feedforward path:

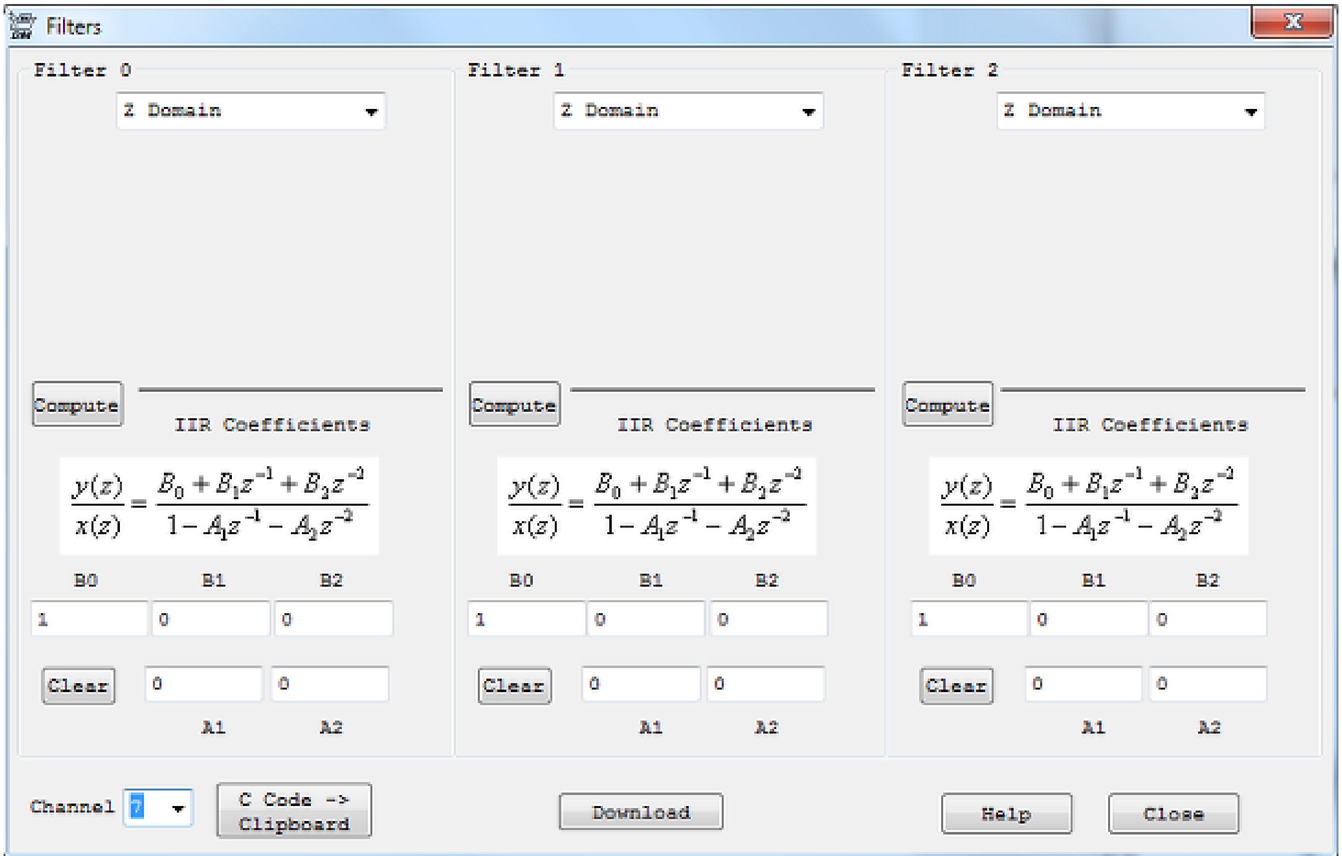


The Screens below show such a configuration. This configuration can be loaded from *AxisAsDAC7.mot*

No Input, DAC Output, Negative Gain, Infinite Following error:

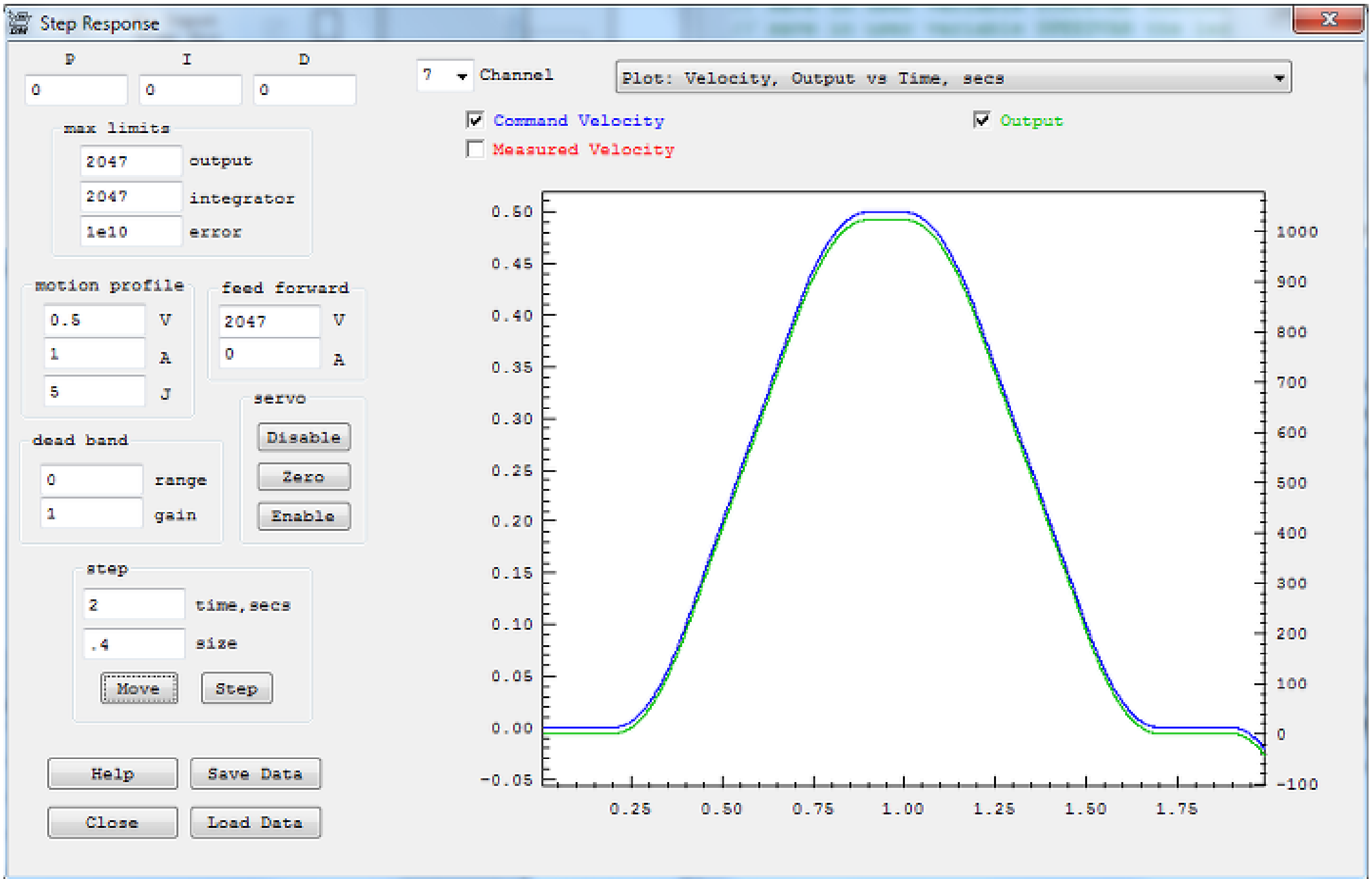


Filters all cleared to have Flat DC Gain=1



PID Zero, Max Output = full DAC Range of 2047 counts, Max Error Infinity, Feed Forward 2047 so that a velocity of 1 count/sec will generate full dac output.

Acceleration = 1 count/s^2 so that full speed will ramp up in 1 second, Jerk=5 counts/s^3 (Acceleration is "smoothed" over 1/5th second). This demonstration Move profile moves at 0.5 counts/sec which results in half of the full DAC output range (1023 counts or 5V analog). Note the output (green plot) follows the Commanded Velocity (blue) in perfect proportion. Any Velocity Range could be used because this is only a theoretical velocity profile and can be scaled to any proportion by the Velocity FeedForward. 1.0 counts/s is used to keep the math simple and to allow Jogging continuously for years without exceeding the Max error or Max Following Error limits.



You may also test your Spindle Axis using Console Screen Jog commands. For example Jog7=0.25 should drive the spindle at 25% full speed.

If your Spindle can accept a +/-10V signal to drive both directions then Jog7 = **-**0.25 should drive in the negative direction.

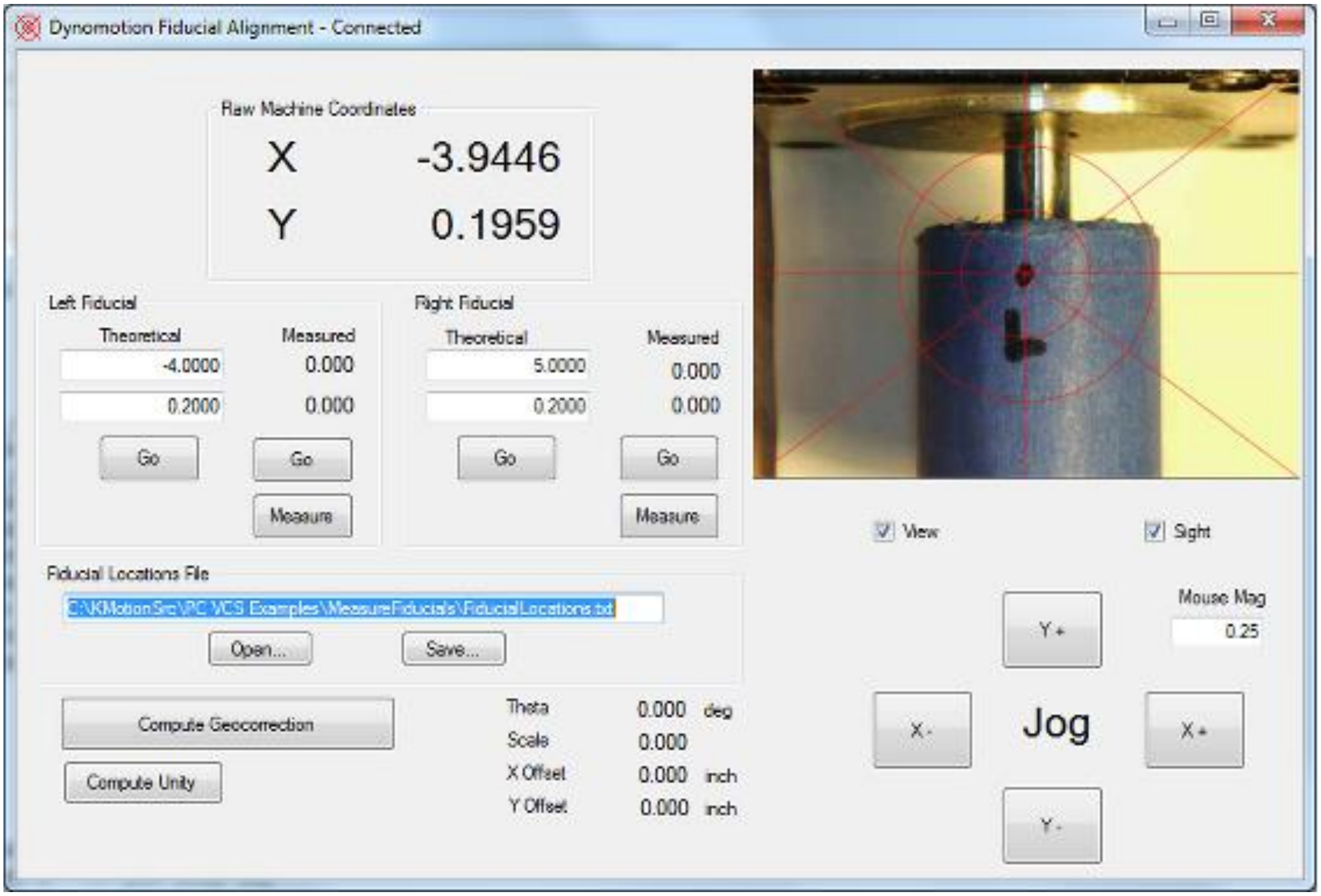
The standard SpindleUsingJogs examples can now be used for an open loop DAC controlled Spindle.

To determine the proper FACTOR in the MySpindleDefs.h file you will need to know what RPM your Spindle runs at when the full 10V analog signal is applied. This will correspond to the RPM where a full Jog Speed of 1.0 will be required. So for example if your RPM reading at 10V is 2500 RPM then configure:

```
#define FACTOR (1.0/2500.0) // to convert RPM to counts/sec
```

KMotionCNC Fiducial Alignment

Fiducial Alignment using the external example C# program <Install>\PC VCS Example\MeasureFiducials

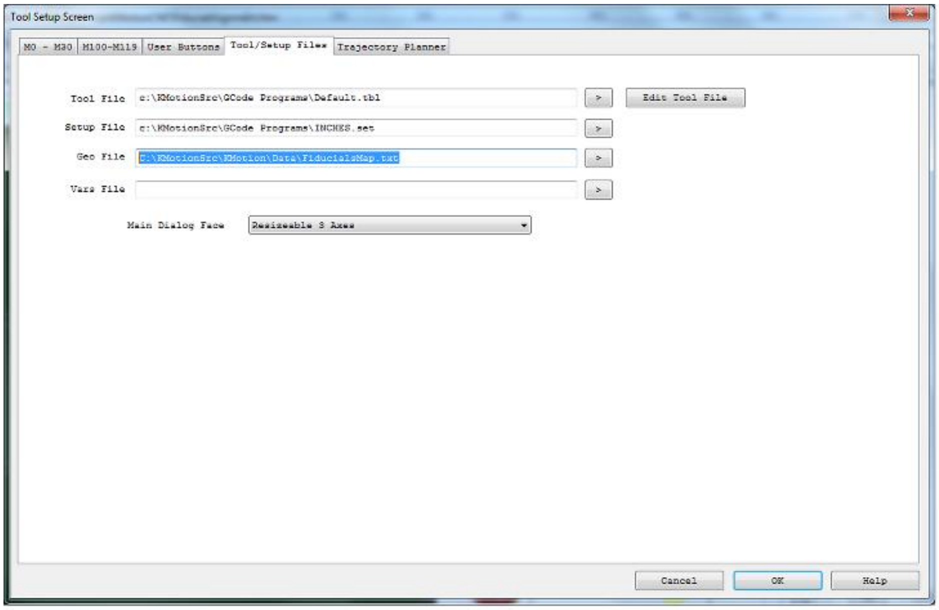


See Video: https://youtu.be/a_p38zktIZI

Measure Fiducials creates and overwrites the Geometric Correction file named:

<Install Directory>\KMotion\Data\FiducialsMap.txt

This should be configured for KMotionCNC to use in the KMotionCNC | Tool Setup | Tool Setup Files configuration page as shown here:



The created Geometric Correction File of 2x2 grid points in the form shown below. The table maps the corrected locations of a simple 1x1 inch square centered on the origin. Note that because the the Geo Correction Transformation extends outside the Grid using the closest Grid's Transformation the size and location of the square is not important.

2,2
1,1
-0.5,-0.5
0,0,-0.411272,-0.508541,0
0,1,0.598128,-0.507930,0
1,0,-0.411883,0.500859,0
1,1,0.597517,0.501470,0

The Measure Fiducials App loads Settings from a file named :

<Install>\\PC VCS Examples\\MeasureFiducials\\MeasureFiducialSettings.txt

Which has this fixed format (avoid extra spaces or tabs). JogSpeed controls the Jog button speeds in counts/inch for both x and y. Resolution should be set to the same value as in KMotionCNC in counts/inch. If your system has more than one Video Capture Device you can control which is selected with the VideoDevice parameter. Zero would be the first detected. MouseMag adjusts the scale factor between Mouse movement and Axes movement.

JogSpeed 600
Resolution 10000 10000
VideoDevice 0
MouseMag 0.25

KMotionCNC accepts a Window Event to Re-load the current Geometric Correction File. The C# Measure Fiducials Example finds any open KMotionCNC Window with:

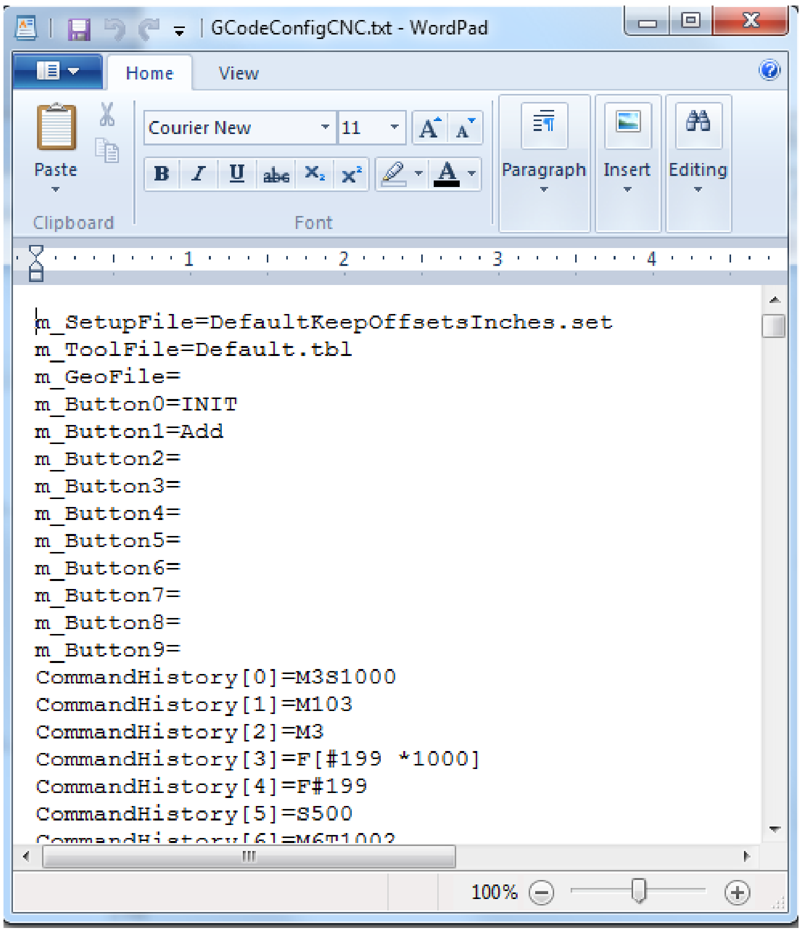
```
[DllImport("User32.dll", SetLastError = true)]
public static extern IntPtr FindWindow(String lpClassName, String
lpWindowName);

IntPtr KMotionCNCWindow = FindWindow("KMotionCNC", null);
```

and sends the Window Event to KMotionCNC with:

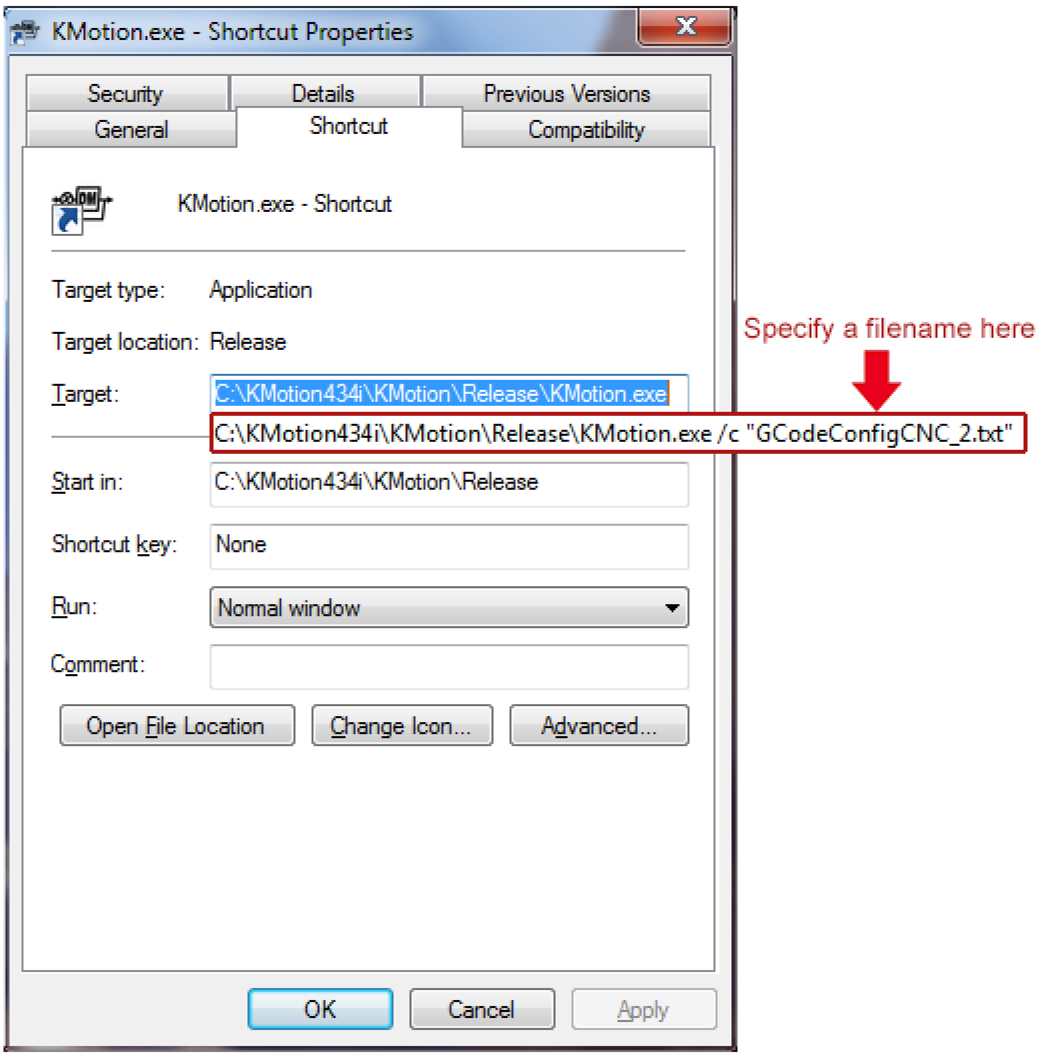
```
[return: MarshalAs(UnmanagedType.Bool)]
[DllImport("user32.dll", SetLastError = true)]
public static extern bool PostMessage(IntPtr hWnd, uint Msg, int wParam,
int lParam);

uint WM_COMMAND = 0x0111;
int ID_ReloadGeoCorrection = 33016;
PostMessage(KMotionCNCWindow, WM_COMMAND, ID_ReloadGeoCorrection, 0);
```

Using Multiple Configurations

Users have the option of saving multiple configurations of KMotionCNC. The entire configuration of KMotionCNC (including all the tool setup files etc.) is stored in a single text file, by default called "**GCodeConfigCNC.txt**", which is located in the \KMotion\Data directory. You can create multiple configurations and save them under different file names. See the configuration file opened in WordPad below:

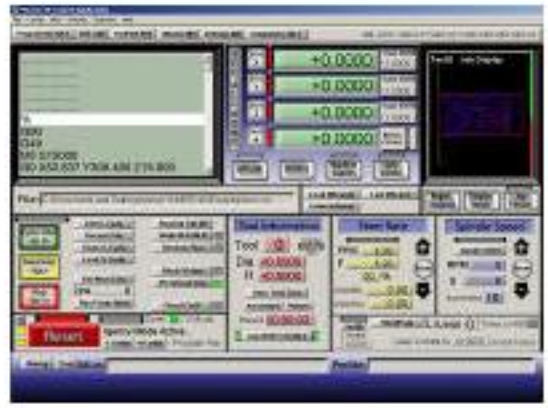


It is possible to create a Desktop icon to launch KMotionCNC using an alternate configuration file. First, create a shortcut to the new configuration by going to the \KMotion\Release directory and right-mouse-clicking on "**KMotion.exe**", and select "**Send to**" and then "**Desktop (create shortcut)**". Once the shortcut has been created, right-mouse click on it, select "**Properties**" and add the configuration file by adding /c "Filename" where "Filename" is the name of the configuration file to be used. Quotes are required if the filename contains spaces. Specifiy a filename which will be in the <Install>\KMotion\Data directory.



Mach3 Plugin

+



KFLOP/Kogna + Mach3

Mach3 is a popular CNC program available for purchase through ArtSoftControls.

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 ***Kogna*** 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

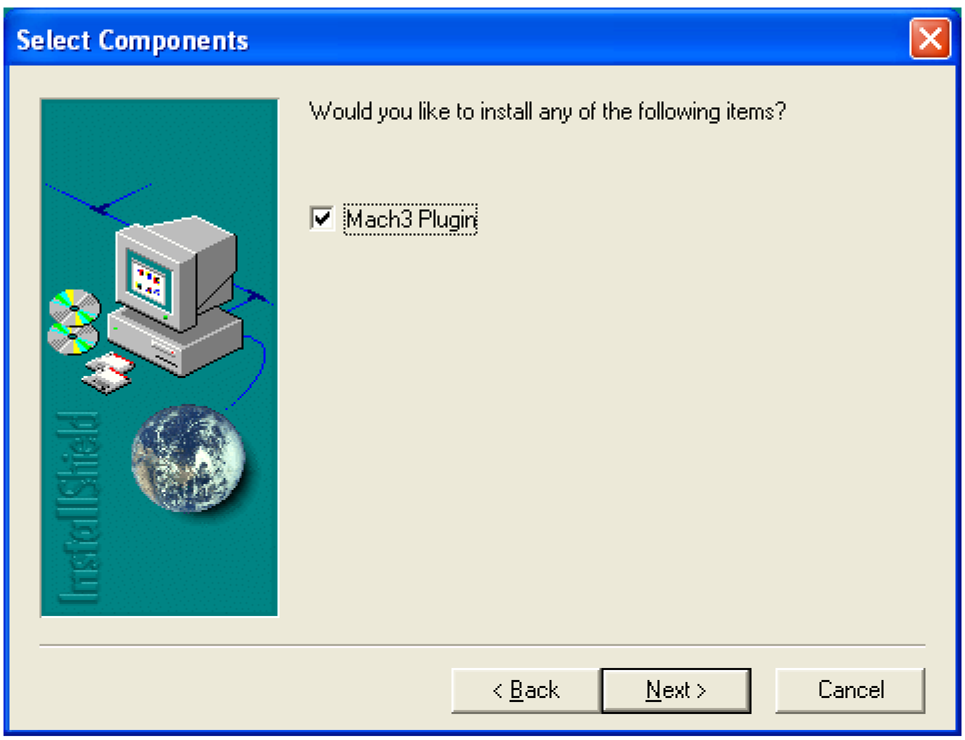
It is not necessary to load the Mach3 Kernel driver.

If KMotion has already been installed, it should be re-installed to make the necessary links between the programs.

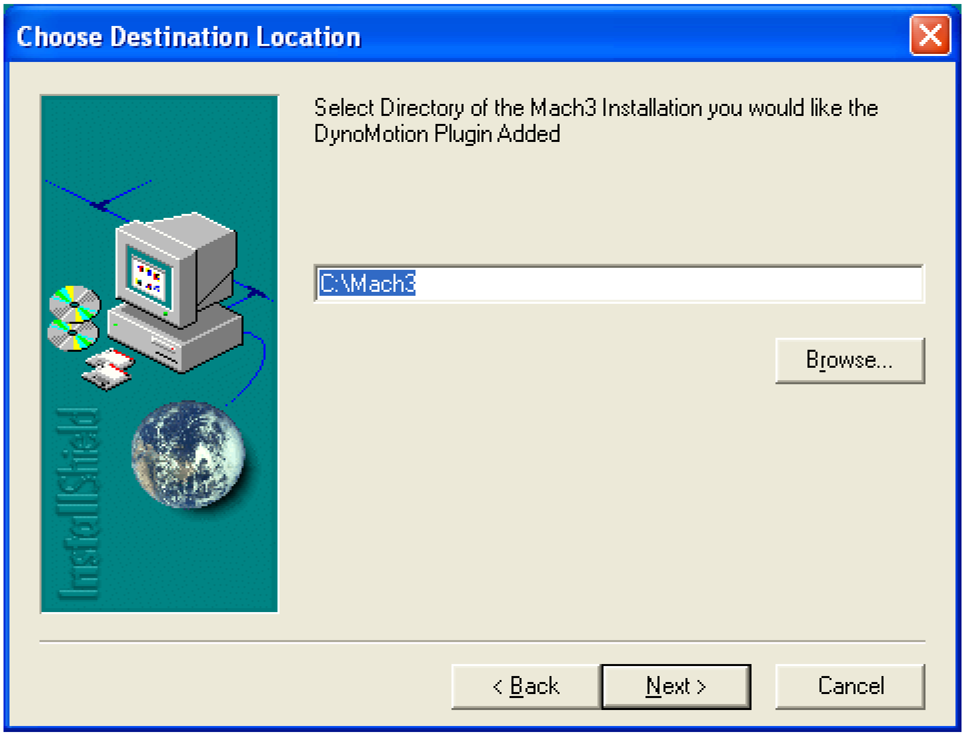
2. Install KMotion

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.

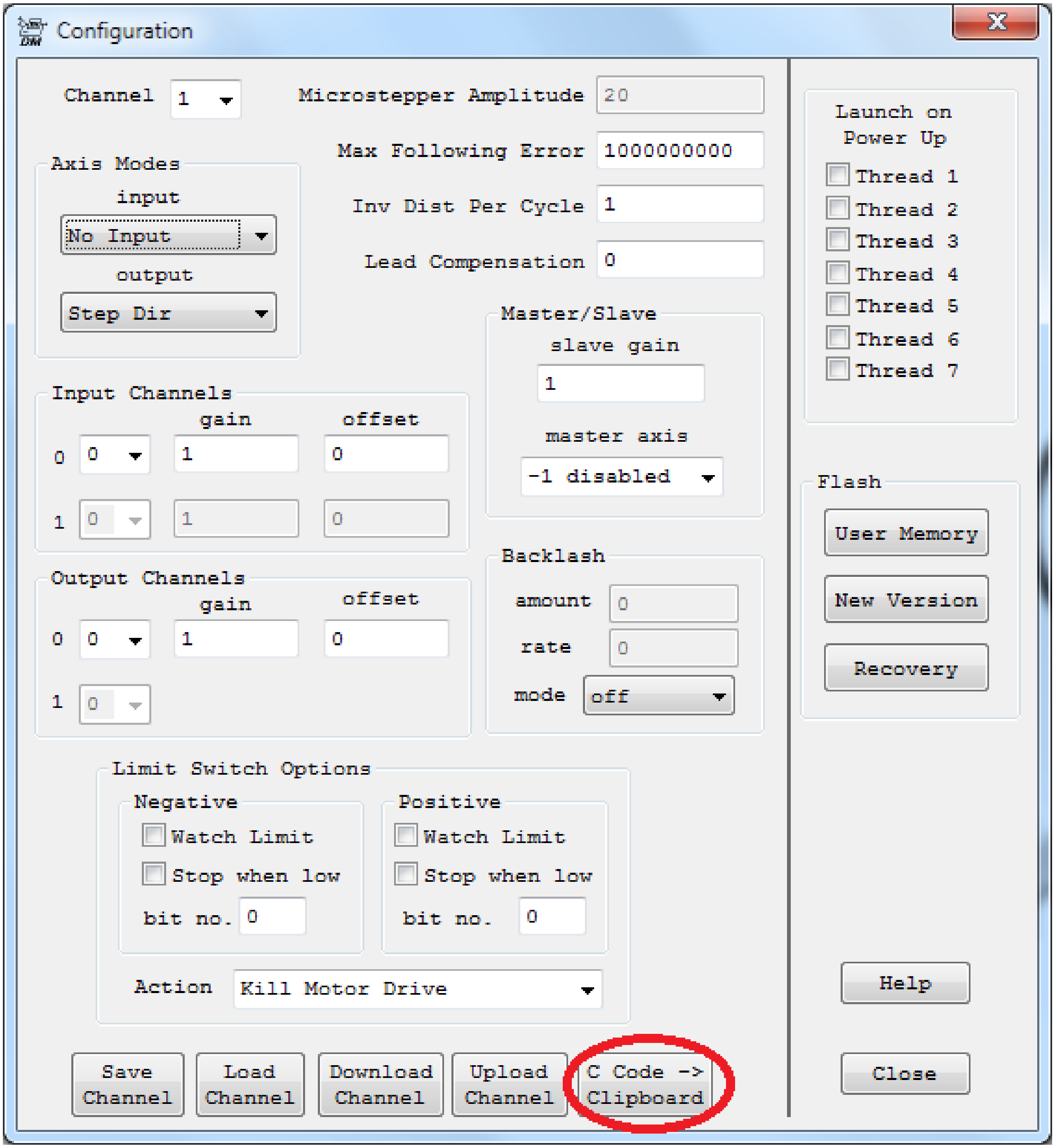
3. Within KMotion configure/tune your motors

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 KFLOP/Kogna 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.

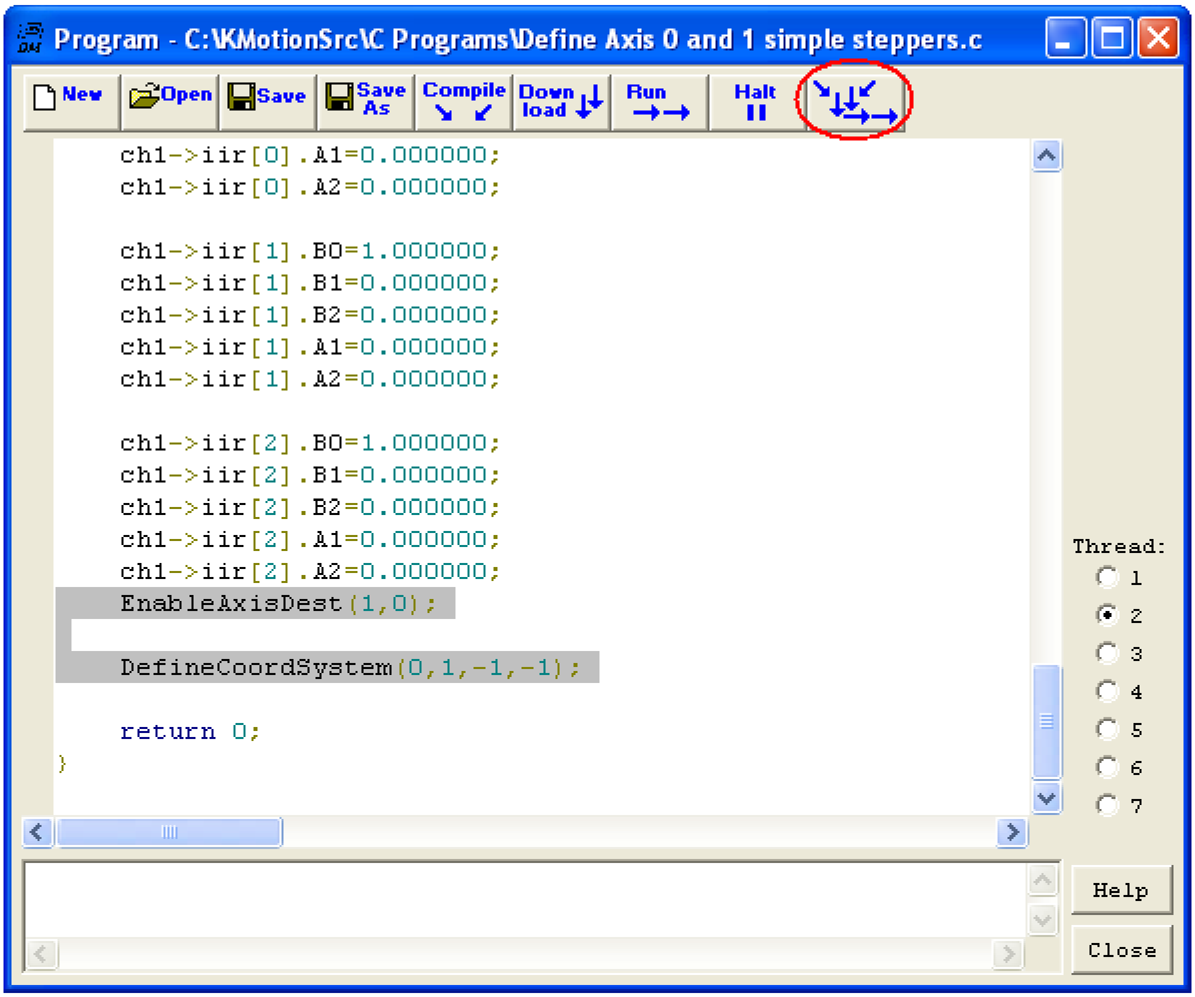
```
ch0->InputMode=ENCODER_MODE;
ch0->OutputMode=MICROSTEP_MODE;
ch0->Vel=100.000000;
```

```
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 KFLOP/Kogna 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".

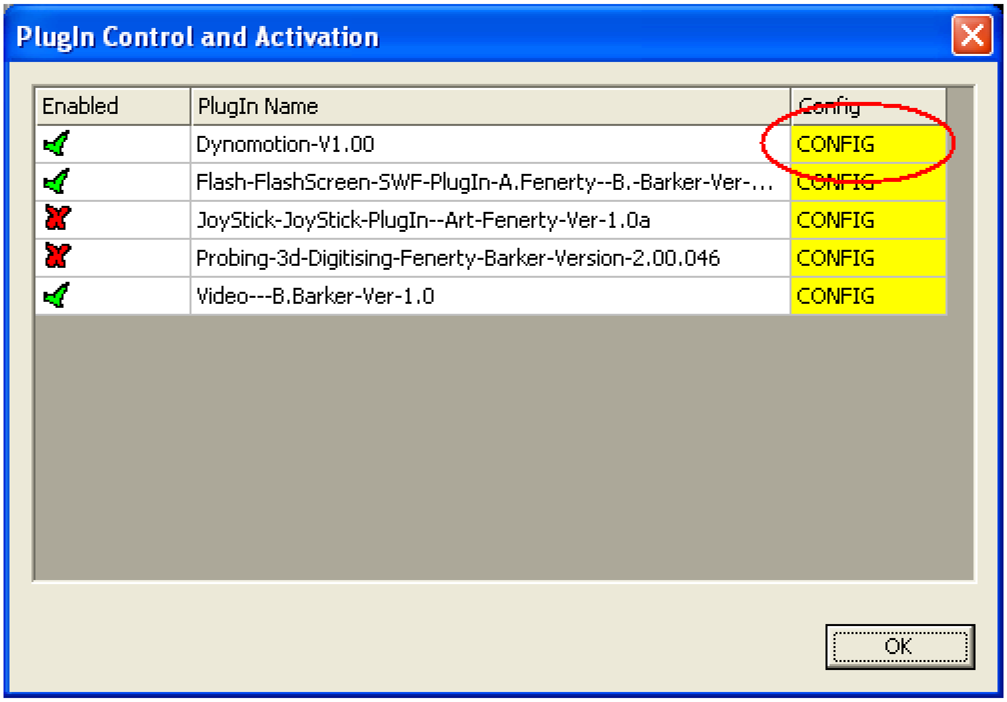
Additional initialization operations may also be added to the C Progam. These might include brushless motor phase finding, homing, activating IO bits, etc..



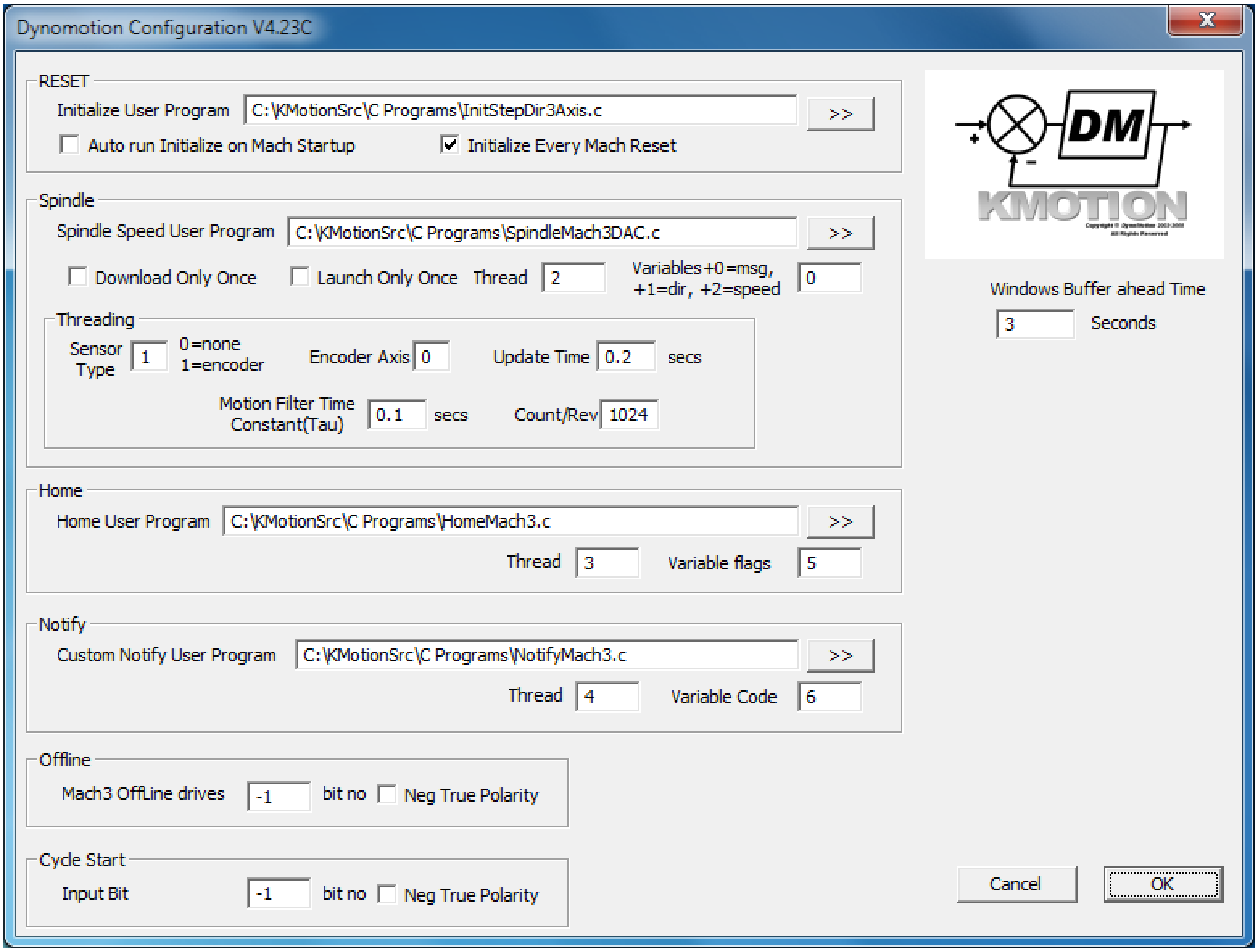
After the Initialization program is finalized and working it should be saved in a known location so that Mach3 may be configured to execute it as described in the next step.

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.



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 KFLOP/Kogna. Message Codes from 10000 to 10999 will be sent to KFLOP/Kogna (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.

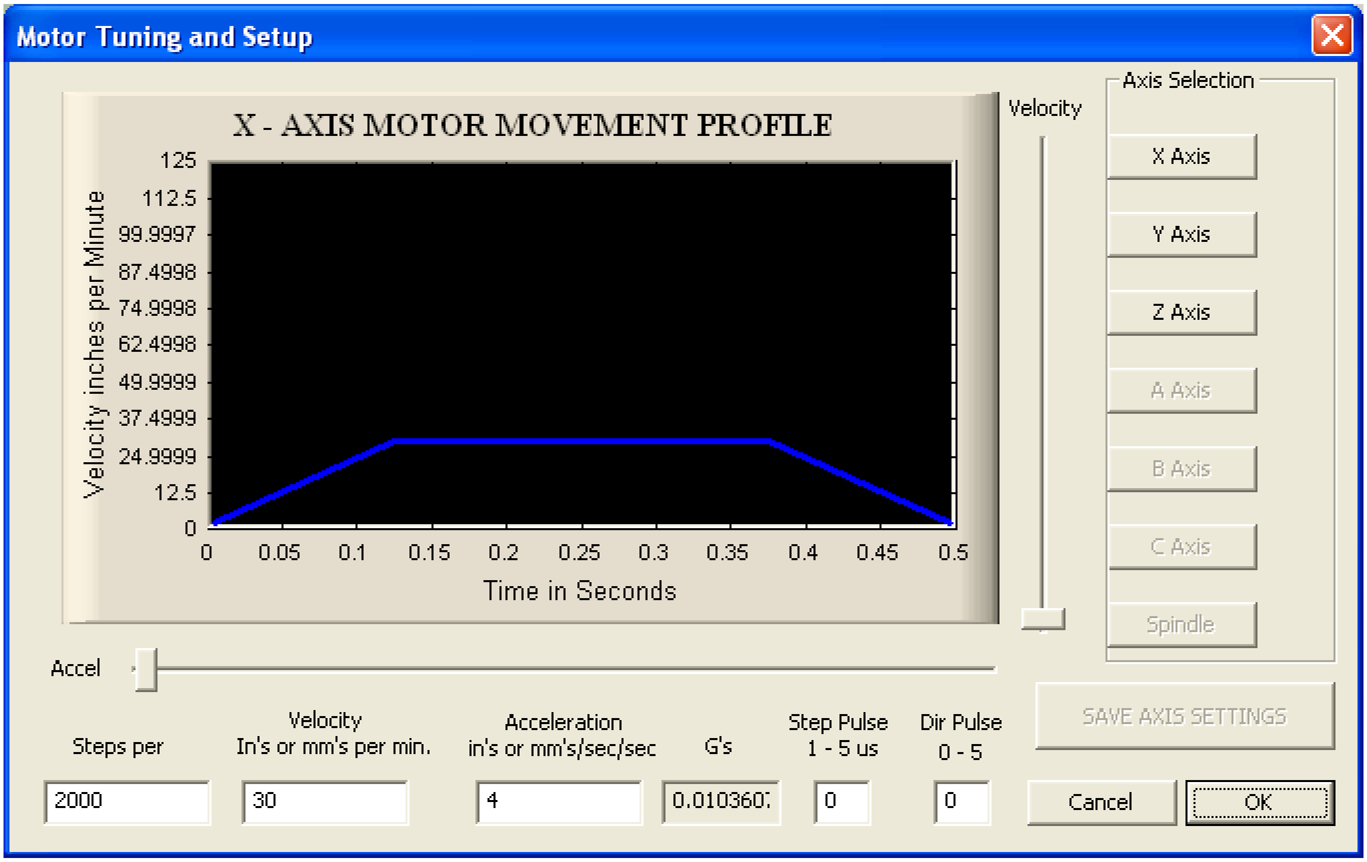
The *Windows Buffer ahead Time* may be changed to keep more or less buffering ahead in the KFLOP/Kogna Board. Using a value too small may cause the buffer to run empty and "starve" for data if Windows becomes non-responsive for longer than that amount of time. Using a value too large will cause feed 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 KFLOP/Kogna Motion Controller than with the original PC Generated Steps. It's usually best to just enter values directly into the cells.

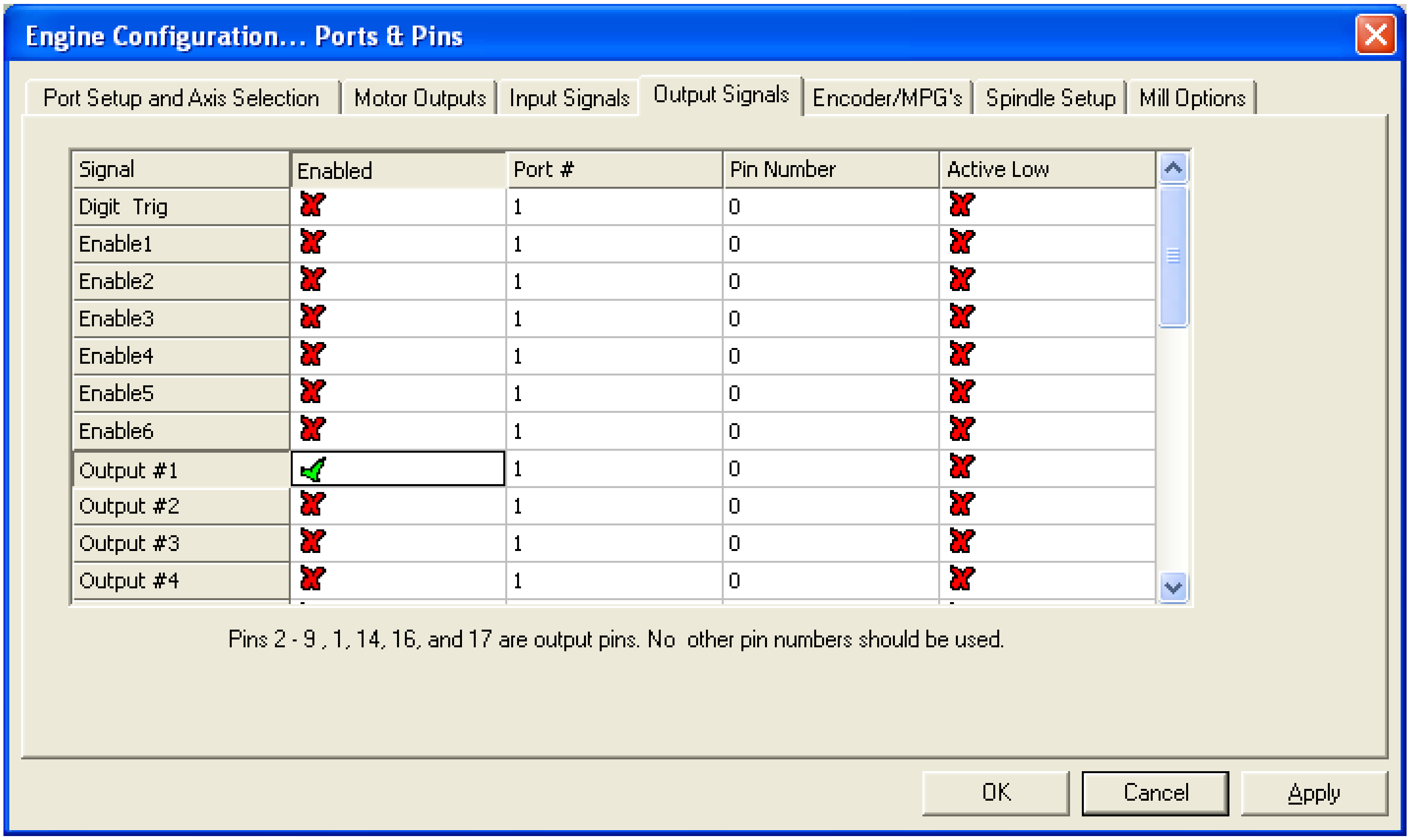


After the parameters have been entered and saved, Press OK to return to the main screen.

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 less than 128 specify **Port#1**. For IO bit numbers 128 or larger, subtract 128 from the bit number and specify **Port #2** instead of Port #1. Extended Virtual IO bits 1024-1151 may also be accessed by specifying **Port #3** and subtracting 1024 from the bit number (note: the first 32 Virtual IO will consume less USB bandwidth because they are uploaded in the KFLOP/Kogna Bulk Status record so use the first 32 if possible).

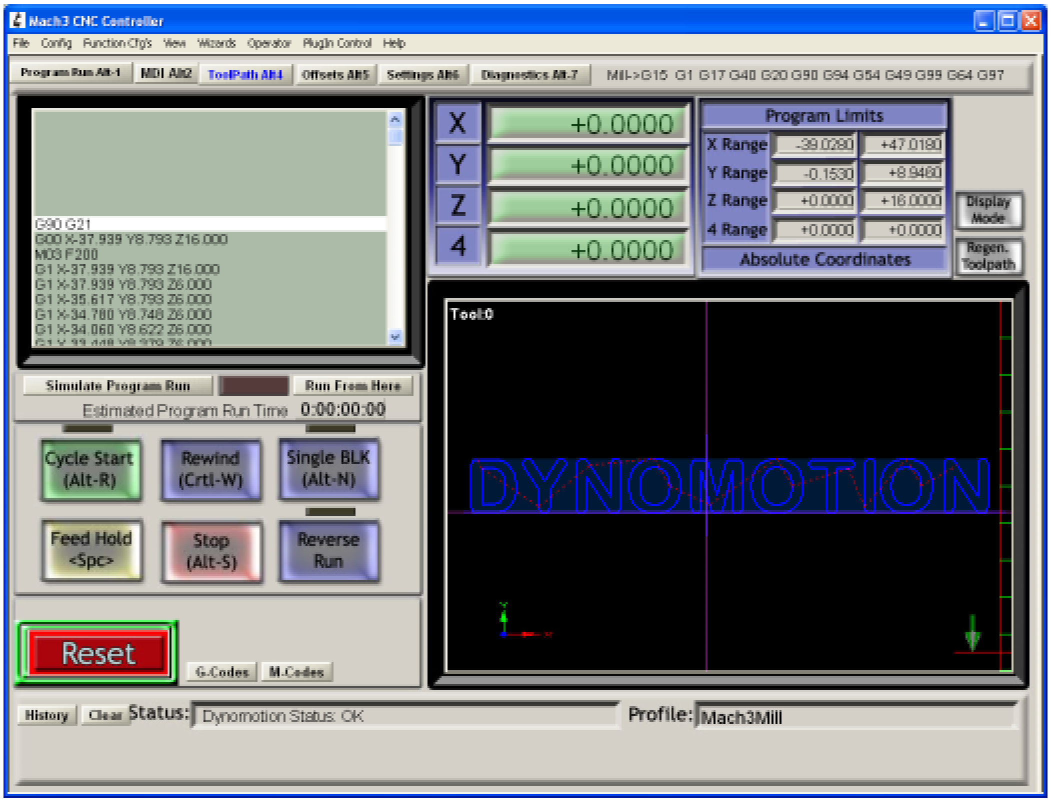
Note: Mach3 typically defaults after an initial install with Output #1 enabled for spindle control on Pin0 (as shown below). Bit 0 is often used on a KMotion board as an Encoder input. Having Mach configure the IO bit as an output will cause the encoder to be inoperable. Disable the output if this is 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.



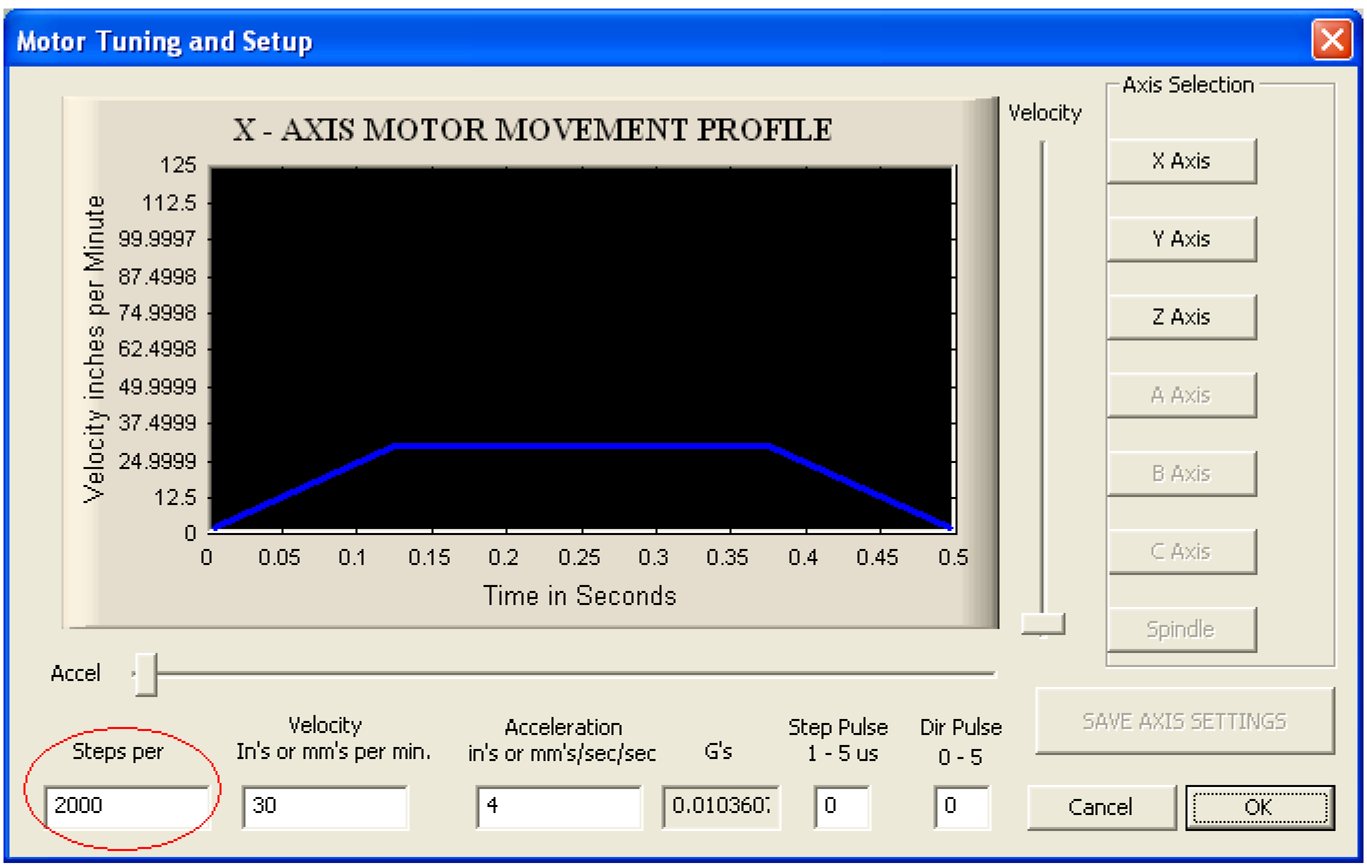
Here the same example GCode file cut (or more accurately burned) into a piece of wood using a Harbor Freight Mini-mill driven with KMotion and Mach3.

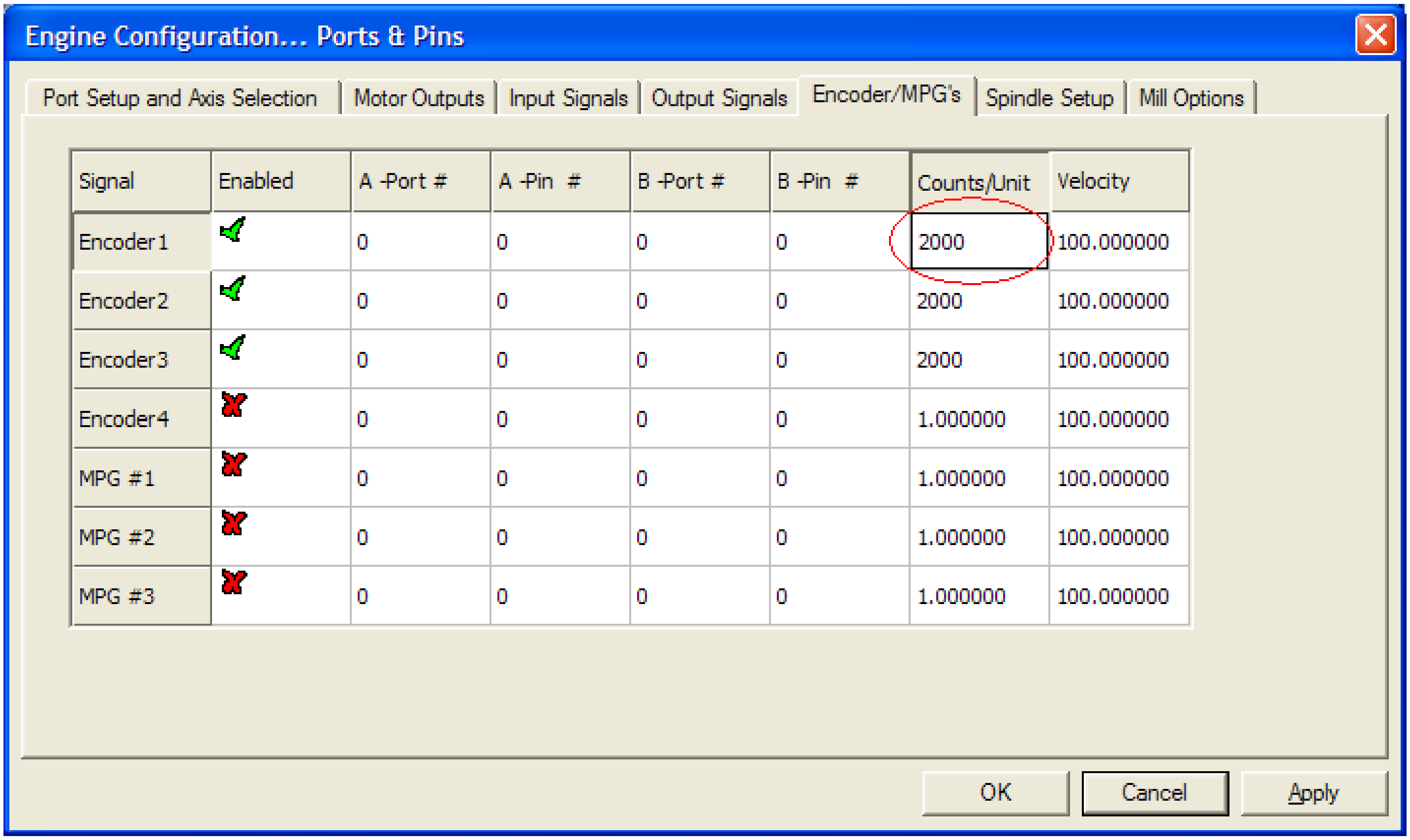
Mach3 Plugin - with Encoders

The following describes the use of linear glass scale encoders or rotary shaft encoders with Mach3. The setup process described apply for KFLOP/Kogna operating in open loop mode with encoders as well as closed loop control.

KFLOP/Kogna 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 µsteps. 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 µSteps) for a Move Plot if properly configured.

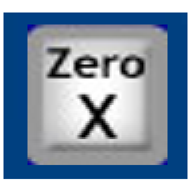
Since the encoder position is already scaled within KFLOP/Kogna to match the µsteps/Unit scale of the motor, the Mach3 Encoder resolution should be set to the same value as the Motor "Tuning" as shown below. The Encoder/MPG screen is opened using the Config|Ports and Pins Menu. The Port and Pin definitions are not relevant when using KFLOP/Kogna and should be set to some unused Port.





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.

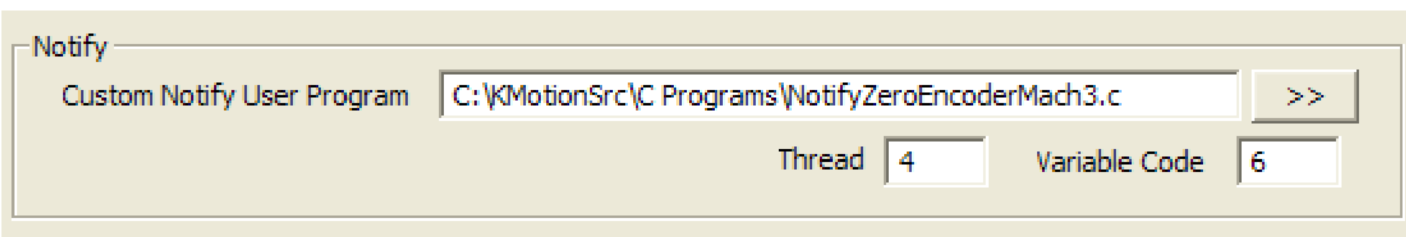



```
NotifyPlugins(10100)      'tell KFLOP to set command to encoder
Sleep 300                 'make sure mach updates
DoOEMButton (1008)        'calculate work offset
```


```
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/Kogna User Program that will process the NotifyPlugin Message Codes to set KFLOP's/Kogna's 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/Kogna User Program via persist.UserData[6]



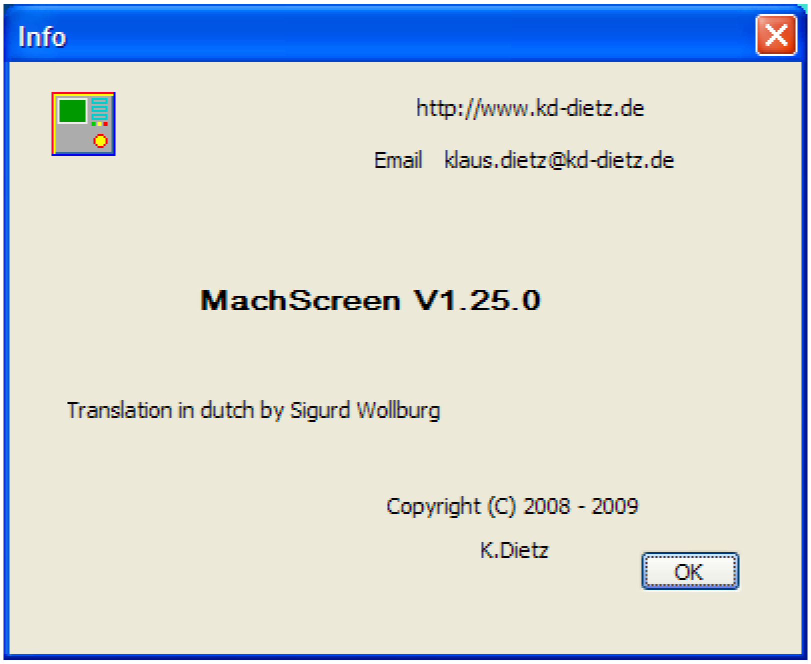
Example File: <Install Dir>\C Programs\NotifyZeroEncoderMach3.c

```
#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);
    }
}
```

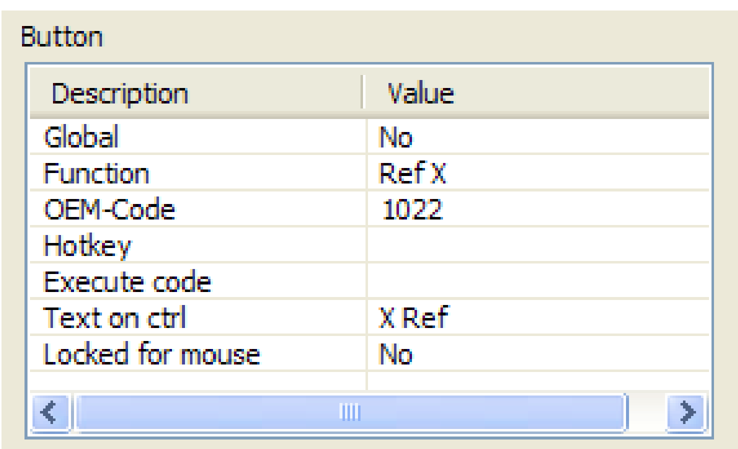
REF Buttons

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

Button

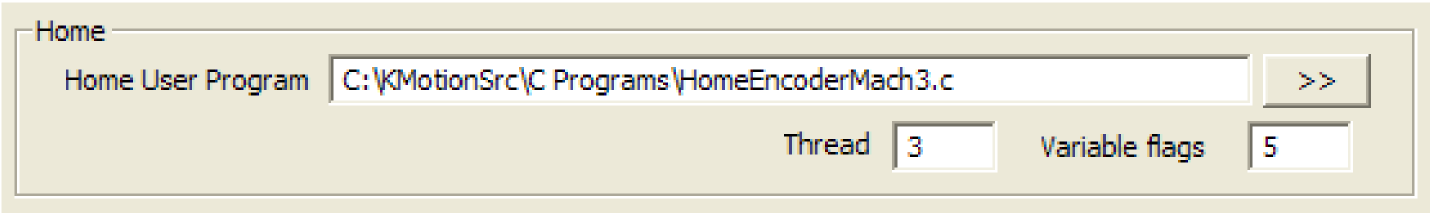
Description	Value
Global	No
Function	Ref Y
OEM-Code	1023
Hotkey	
Execute code	
Text on ctrl	Y Ref
Locked for mouse	No

Button

Description	Value
Global	No
Function	Ref Z
OEM-Code	1024
Hotkey	
Execute code	
Text on ctrl	Z Ref
Locked for mouse	No

The Dynomotion Plugin passes these Home requests to KFLOP/Kogna to handle with a Home User Program. In the **Config|Config Plugins|Dynomotion** set an appropriate KFLOP/Kogna User Home Program. A **flag** variable is also passed to tell which axis is to be homed (Note that the flags is defined to be passed to the KFLOP/Kogna 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.

Note: if Homing Inputs are enabled in Mach3 | Config | Ports and Pins | Input Signals | X Home, Y Home, Z Home, A Home, B Home, C Home then Mach 3 will NOT call the Plugin to do Homing. Please make sure these Inputs are NOT enabled.



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);


	}
}
```

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.

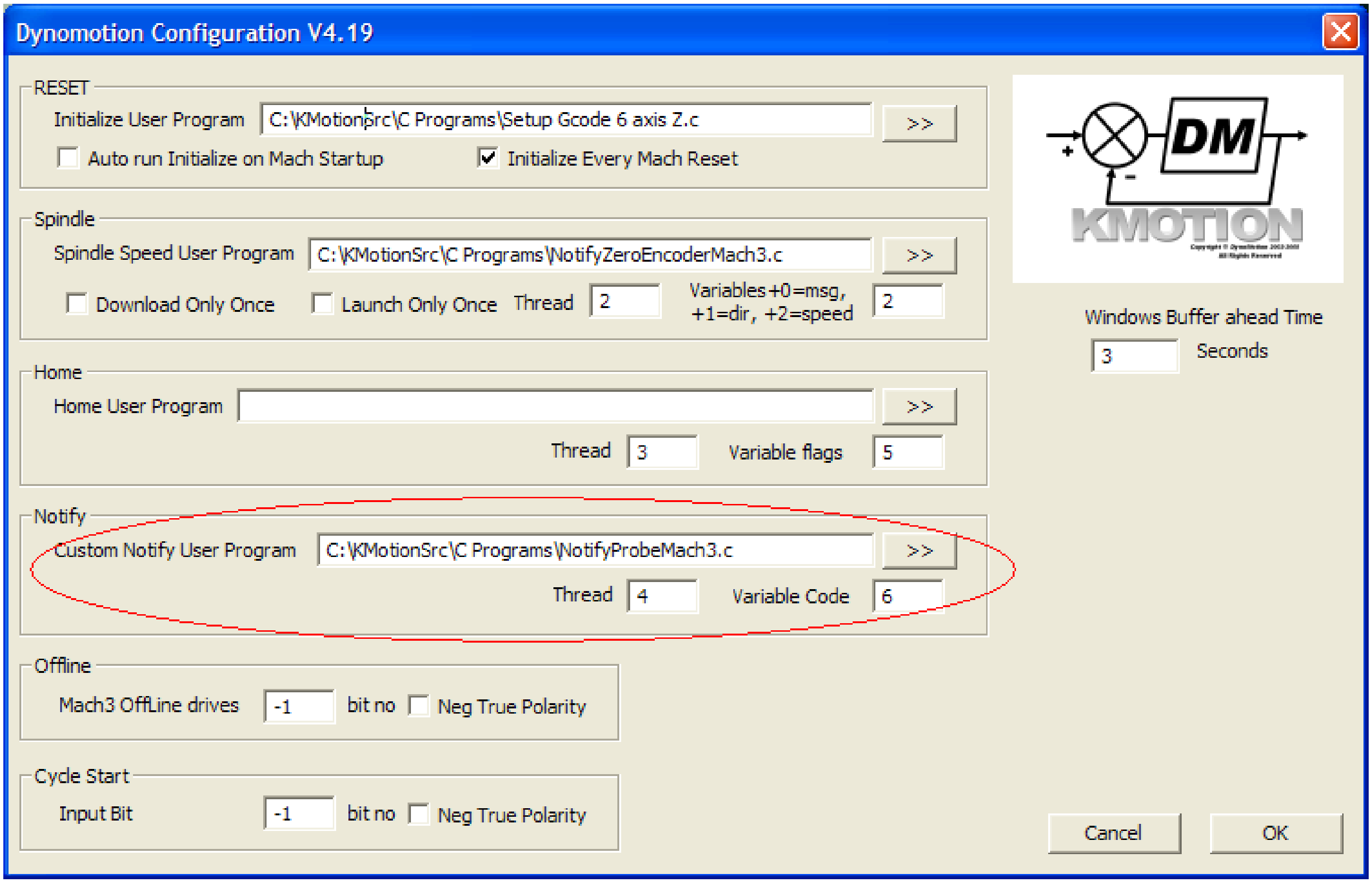
The following sequence might be used to perform a probe and then perform a move relative to the captured Z position.

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 KFLOP/Kogna User Program

In order to perform Probing, 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 KFLOP/Kogna 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

Note: defines below in PC-DSP.h should be used to symbolically reference the persist.UserData Variables

```
#define MACH3_PROBE_STATUS_VAR 62
#define MACH3_PROBE_RESULTS_VAR 50
```

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
	// flag is 3 - Tells Plugin to upload status (3) to
	//             DRO 1100 and let User handle the error
	//
	// 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
	#define PROBE_ERROR_HANDLING 0  // 0 Stops Mach3 on probe error
//	#define PROBE_ERROR_HANDLING 3  // 3 User must query DRO 1100 and 
handle error

	if (msg==20000)
	{
		double *d = (double 
*)&persist.UserData[MACH3_PROBE_RESULTS_VAR];
		int flag=1;
		
		persist.UserData[MACH3_PROBE_STATUS_VAR]=PROBE_ERROR_HANDLING
;

		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[MACH3_PROBE_STATUS_VAR]=flag;
            StopCoordinatedMotion();
      }
}
```

Mach3 Plugin - Passing DROs

Mechanism for transferring values back and forth between Mach3 and KFLOP/Kogna

Mach3 User DROs 1 to 50 map to KFLOP/Kogna UserData 0 to 99 (2 words each double)

To Read from KFLOP/Kogna to Mach3 use NotifyPlugins codes 18001 to 18050

To Write from Mach3 to KFLOP/Kogna 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/Kogna, notifies KFLOP/Kogna to perform the Tap Cycle, then waits until KFLOP/Kogna 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/Kogna 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/Kogna UserData variables, Triggers KFLOP/Kogna to perform the Tap Cycle, then waits until KFLOP/Kogna 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/Kogna 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/Kogna (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);
```

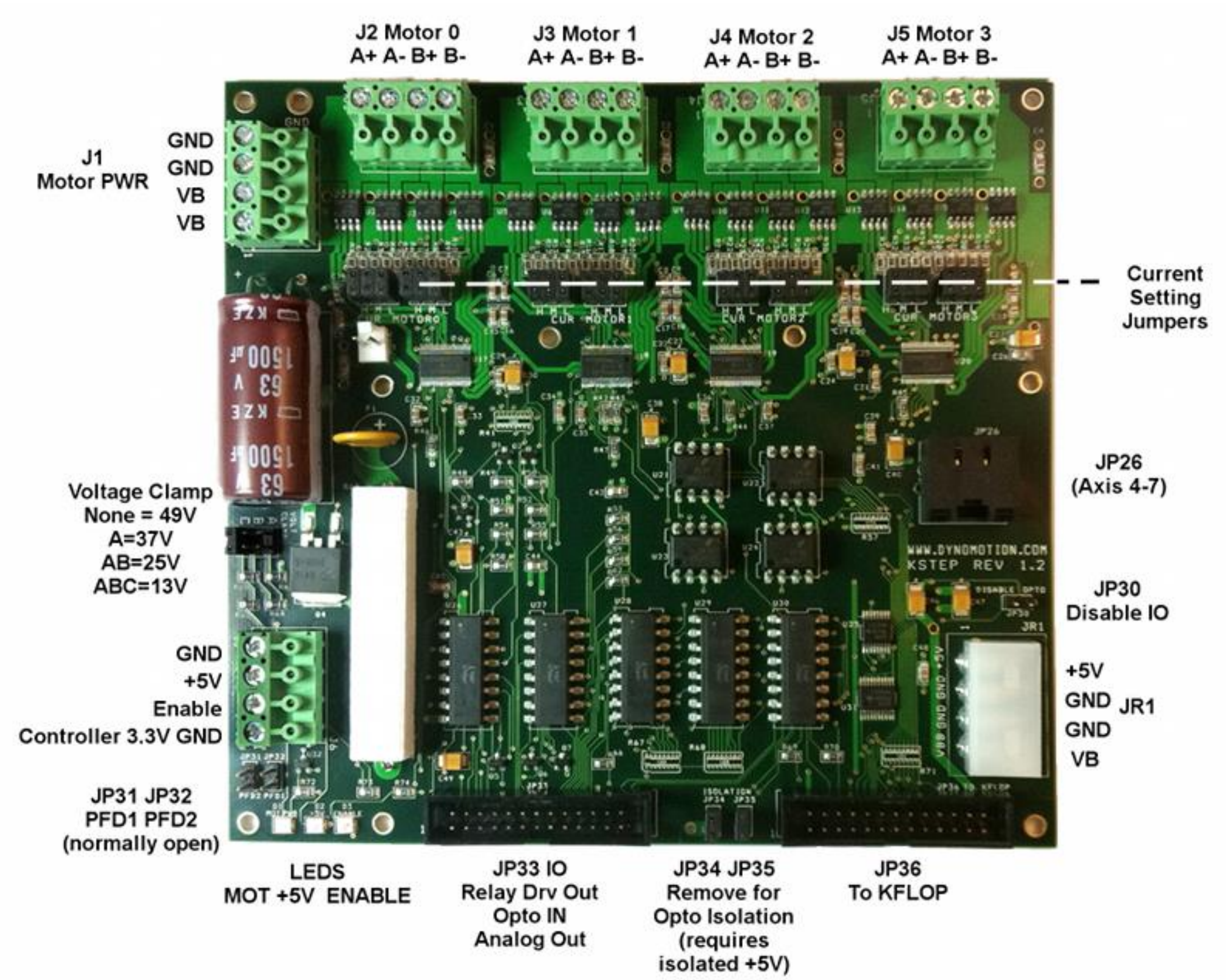
```
	double RetractRate  = RetractRPM/(TPI*60.0);

	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();
}
```

KStep 1.2 Hardware

Function	Parameter	Specification
Micro Stepper Drivers	Type Number Microstepping Supply Voltage Motor Coil Current Max Step Rate	Bipolar Dual Full Bridge Chopper Drivers 4 16X 12-48VDC Common Jumper Programmable 8 levels (0.6 - 5A) 500KHz
Logic Inputs (Step,Dir,Enable,Relay PWM,MuxCtrl)	Voltage Current	3.3V LVTTL (2.4V min High 0.4V max low) 16ma max
Analog	Number Input Supply Range Output Impedance Response, Tau	1 fully Isolated 5-30V <1K ohm 10ms
Inputs	Type Number Internal Series Resistor Min On Voltage Max Allowed Voltage Max Off Voltage	Opto Isolated Common Anode 16 10K ohm 11V 25V 1.0V
Outputs	Type Number Type Max Voltage Max Current	Opto Isolated 2 Opto Isolated Darlingtion 30V 100ma
Voltage Clamp	Power peak Power Average (3 sec) Levels	1KW 10W Jumper Programmable (13V, 25V,37V,49V)
5V Logic Supply	Voltage Max Current	+5V ± 5% 0.05 A
3.3V Logic Supply	Voltage Max Current	+3.3V ± 5% 0.05 A
Terminal Strips	Screw Terminals	24 Pluggable Screw Terminals (6) x 4 pin 5mm pitch
Environment	Operating Temperature Storage Temperature Humidity	0-40º C 0-40º C 20-90% Relative Humidity, non-condensing
Dimensions	Length Width Height	5.5 inches (140mm) 6.0 inches (151 mm) 1.0inches (24 mm)
Green	RoHS	Compliant

KStep – Connector Pinouts



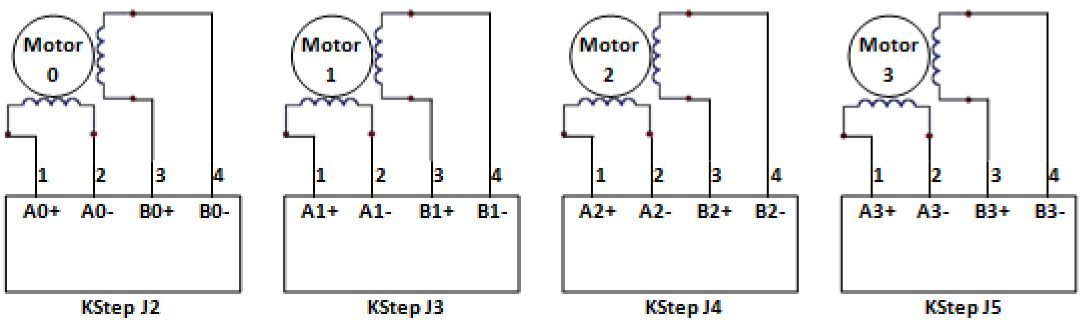
J1 Motor Power

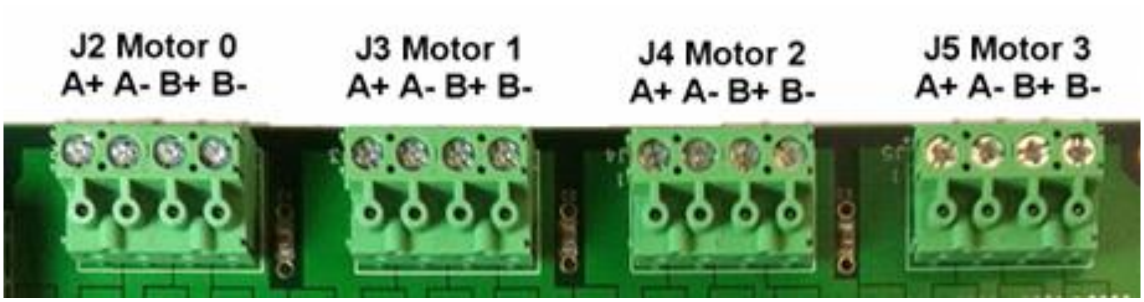
Motor power VBB from 12-48V may be applied through J1 or alternately through JR1. For applications that require more than 10A of supply current both of the GND and VBB connections should be used.

Caution Voltage Clamp jumpers must be set to a Higher Voltage than the applied voltage.

Caution if JR1 is connected to a PC Power Supply 12V will be applied to VB

J2 J3 J4 J5

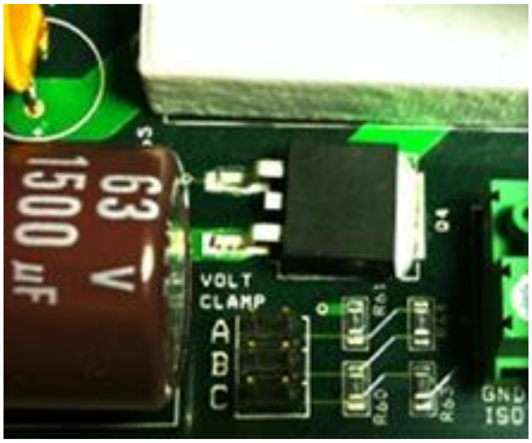




Connect Stepper Motors to connectors J2 through J5. A Stepper Motor has two independent coils A and B. This should be verified with an Ohmmeter. Connect one motor coil across the A terminals and the other coil across the B terminals.

Caution cross wiring the motor terminals is likely to cause damage to KStep and/or the motor.

Voltage Clamping

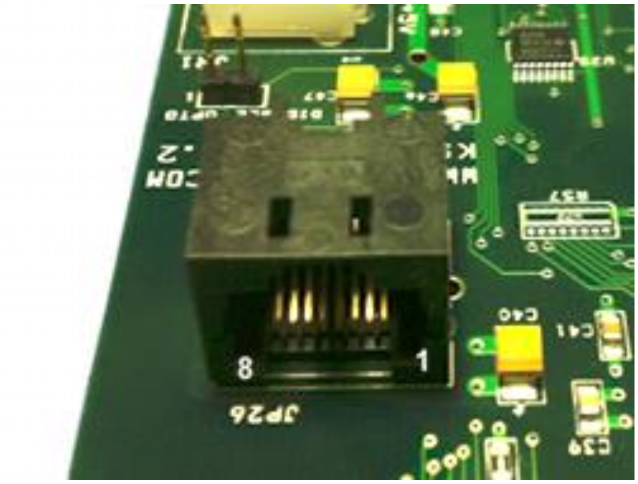


KStep features a jumper programmable Voltage Clamping Circuit. Voltage spikes due to Regenerative Braking and other sources can potentially raise the supply voltage up to a level that can damage electronics including KStep or the power supply. Voltage Spikes may also cause a power supply to trip a fault and shut down. KStep's Voltage clamp can be used to avoid these problems. KStep's Voltage Clamp can be programmed for 4 different voltage levels. It is not possible to disable the Voltage Clamp but the clamping voltage level may be changed. With all jumpers removed the Voltage will be clamped at 49V. Each jumper added will reduce the Clamping voltage by 12V. The order of the jumpers is not important as each reduces the Voltage by the same amount (12V). This allows common supply voltages of 12, 24, 36, and 48V to be used.

Caution connecting a supply voltage higher than the clamping voltage will draw excessive current and is likely to cause damage to KStep or the Power Supply.

Jumpers Installed	Example	Clamping Level
None	None	49V
1	A	37V
2	A+B	25V
3	A+B+C	13V

JP26



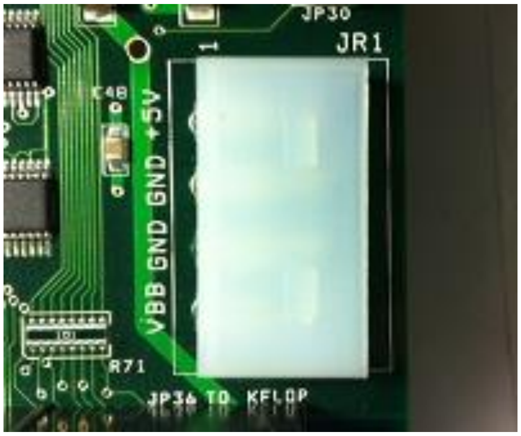
KStep can accept Step/Dir signals in from either JP26 or JP36 whichever is more convenient. Normally if two KSteps are connected to KFLOP/Kogna in order to have 8 axes in the system then JP26 will be used for the 2nd KStep. KFLOP/Kogna drives Step/Dir Generators 4 - 7 on its JP5 connector which when connected to KStep's JP26 connector will drive Motors 0 - 3. When driving signals into JP26 GND, +5V and Enable must be provided into KStep in some manner. J6 is usually the simplest but JR1 and JP36 are also possibilities.

Pin	KFLOP/Kogna Name	KFLOP/Kogna Output	KStep Input
1	IO36	Step 4	Step 0
2	IO37	Dir 4	Dir 0
3	IO38	Step 5	Step 1
4	IO39	Dir 5	Dir 1
5	IO40	Step 6	Step 2
6	IO41	Dir 6	Dir 2
7	IO42	Step 7	Step 3
8	IO43	Dir 7	Dir 3

Disable Optos Jumper

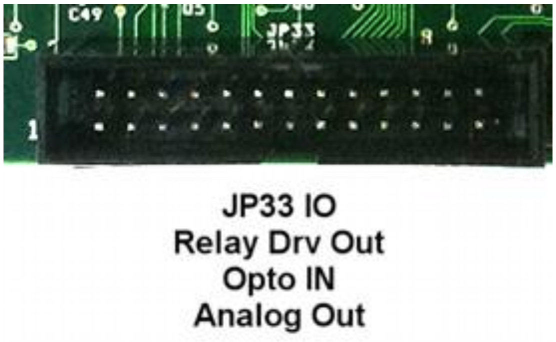
KStep normally drives JP36 pins 11 - 14 with selected Opto Input Data. If Opto Input Data is not desired this jumper disables this feature and leaves these pins in a high impedance state. This might be necessary if the connection to KFLOP/Kogna desires to use the associated pins for some other purpose such as KFLOP/Kogna Encoder inputs #2 and #3.

JR1

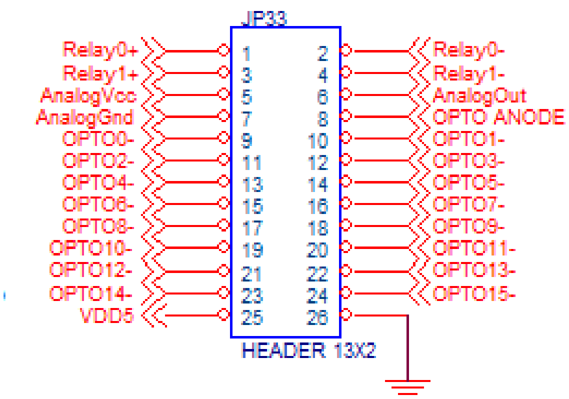


JR1 is a molex Disk drive type connector that can be used to provide Motor power (VBB) and +5V to KStep. Motor power is common to all 4 Motor Drives and may be applied either through JR1 or J1 screw terminals. A PC Power Supply often provides +5V on pin 4 and +12V on Pin 1 so caution should be made to assure different power supply voltages are not connected to both JR1 and J1. The +5V connection is used to power the Motor driver circuits. The +5V and GND connections are to the motor side of the isolation and will be isolated from the KFLOP/Kogna side if the Isolation Jumpers are removed.

JP33



KStep adds 2 opto isolated relay drivers (100ma @ 36V), one Analog Output, and 16 opto isolated 12-24V inputs. All these IO are optically isolated regardless of the Isolation Jumpers set for the Isolation on the Step/Dir and enable signals. Opto Inputs have a common Anode normally connected to either +12V or +24V. Shorting any of the OPTO- pins to GND will activate the input. The Opto Inputs have a 10K ohm series resistance.

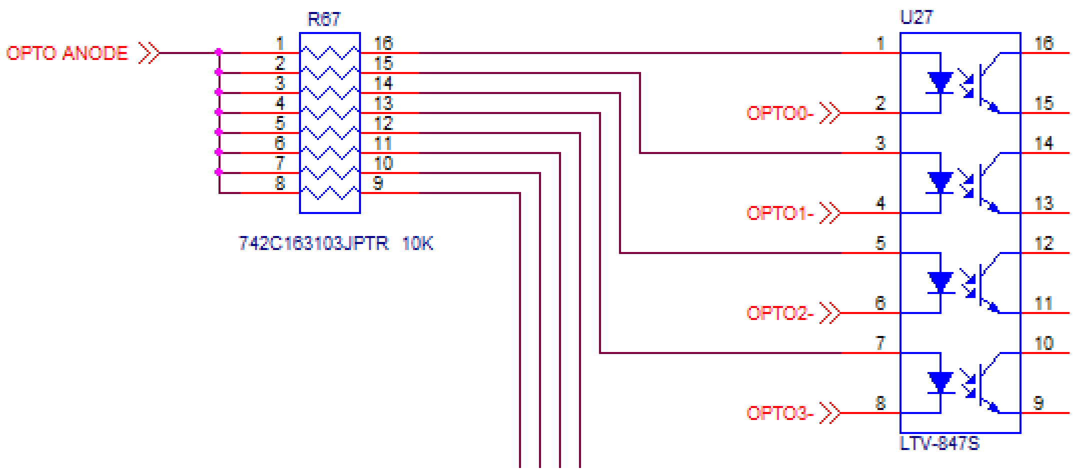


JP33 Pin	KStep Signal	KFLOP/Kogna Related Signal
1	Relay0+	IO 0 Output
2	Relay0-	IO 0 Output
3	Relay1+	IO 1 Output
4	Relay1-	IO 1 Output
5	Analog Vcc	

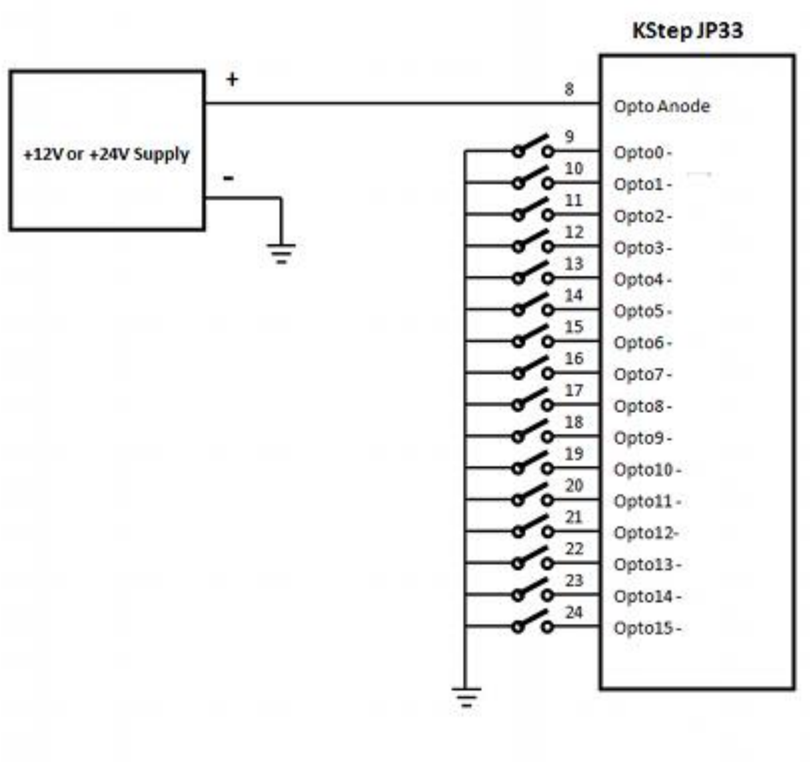
6	Analog Out	IO 44 PWM Output
7	Analog Gnd	
8	Opto Anode	
9	Opto IN 0	IO 168 Input
10	Opto IN 1	IO 169 Input
11	Opto IN 2	IO 170 Input
12	Opto IN 3	IO 171 Input
13	Opto IN 4	IO 172 Input
14	Opto IN 5	IO 173 Input
15	Opto IN 6	IO 174 Input
16	Opto IN 7	IO 175 Input
17	Opto IN 8	IO 176 Input
18	Opto IN 9	IO 177 Input
19	Opto IN 10	IO 178 Input
20	Opto IN 11	IO 179 Input
21	Opto IN 12	IO 180 Input
22	Opto IN 13	IO 181 Input
23	Opto IN 14	IO 182 Input
24	Opto IN 15	IO 183 Input
25	Vdd +5V	+5V KFLOP (if non isolated)
26	GND KStep	GND KFLOP (if non isolated)

Optically isolated Inputs

The input circuit for 4 of the 16 opto inputs is shown below. Note there is a common Anode for all 16 opto inputs which is normally tied to a +12 or +24V supply. Each of the negative opto pins are then switched to ground (return for the +12 or +24V supply).

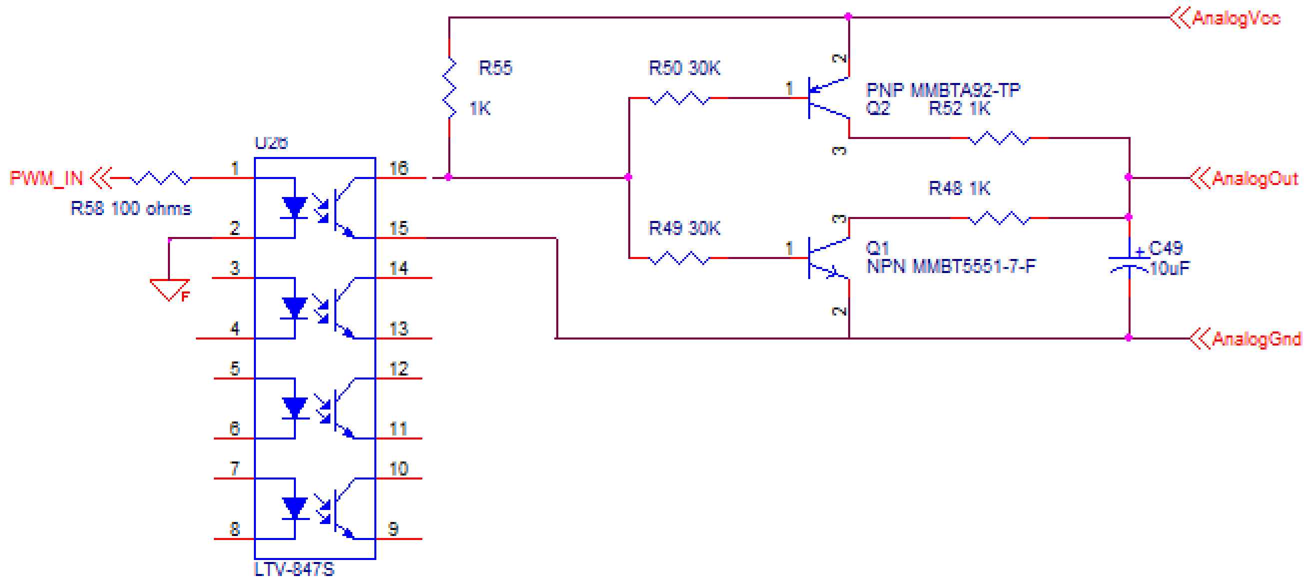


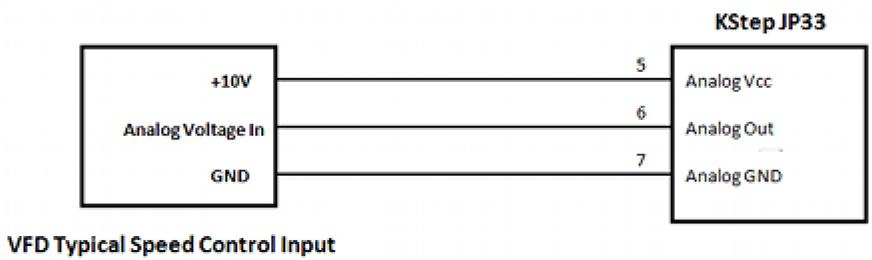
External Wiring would typically be arranged such as:



Optically isolated PWM to Analog Circuit

A 3.3V PWM output from KFLOP/Kogna IO 44 JP7 pin 5 is normally connected to KStep JP36 Pin 5 to drive the PWM to Analog circuitry. Often a VFD will have 3 input terminals where a potentiometer can be connected as speed control. The three connections AnalogVcc, AnalogOut, and AnalogGnd can be connected to perform a similar function as a potentiometer. The AnalogOut voltage will vary from AnalogGnd to AnalogVcc voltage as a function of the PWM duty cycle.

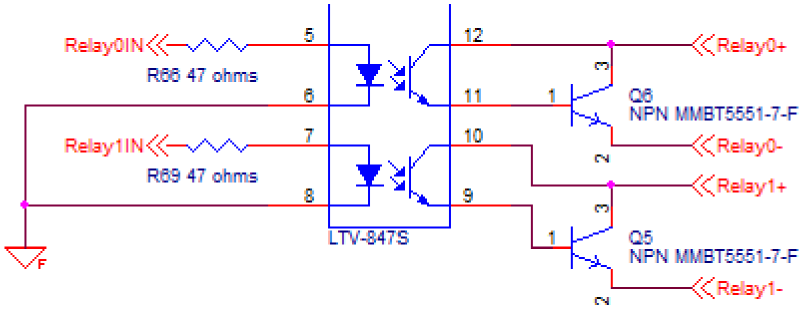




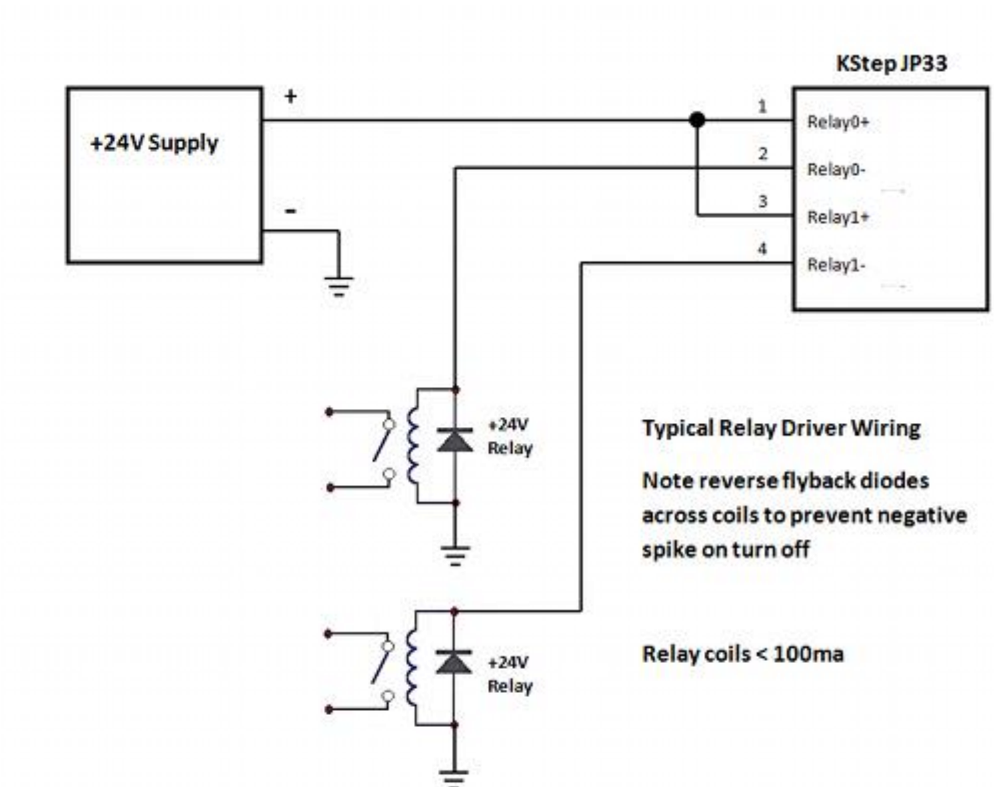
Relay Driver Outputs

Two Optically Isolated Outputs are provided to drive medium power devices such as relay coils. Loads up to 30V @100ma may be driven.

The internal KStep circuitry is shown below which converts the incoming 3.3V LVTTL inputs to darlington transistor outputs.



A typical wiring diagram driving 24V relays.



JP31JP32 PFD1 PFD2 Jumpers

These Jumpers can be used to control the Fast/Slow/Mixed delay settings of all 4 Motor Drives. In almost all cases these jumpers should remain removed for best performance.

Current Setting Jumpers

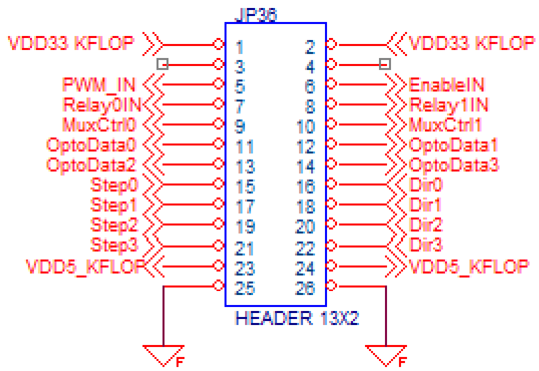
Current for each Stepper Coil is set with a set of 3 jumpers labeled H M L (high, medium, low).

Both sets of 3 jumpers for each motor coils must be set to the same level.

H	M	L	Current (Amps)
Removed	Removed	Removed	0.63
Removed	Removed	Installed	1.25
Removed	Installed	Removed	1.88
Removed	Installed	Installed	2.50
Installed	Removed	Removed	3.13
Installed	Removed	Installed	3.75
Installed	Installed	Removed	4.38
Installed	Installed	Installed	5.00

JP36 To KFLOP/Kogna

JP36 is designed to connect directly to KFLOP's/Kogna's JP7 in a 1:1 connection. KFLOP/Kogna is then able to supply the 4 sets of Step/Dir signals, the Enable, the Analog PWM signal, the 2 Relay Driver outputs, and connections for the 16 opto inputs, as well as +5V (used if running without opto isolation). To conserve KFLOP/Kogna IO pins the 16 opto inputs are read back in a multiplexed fashion 4 bits at a time where 2 outputs from KFLOP/Kogna select which bank of 4 are currently selected. +3.3V is also provided to power the opto multiplexing circuitry.



JP34 JP35 Isolation Jumpers

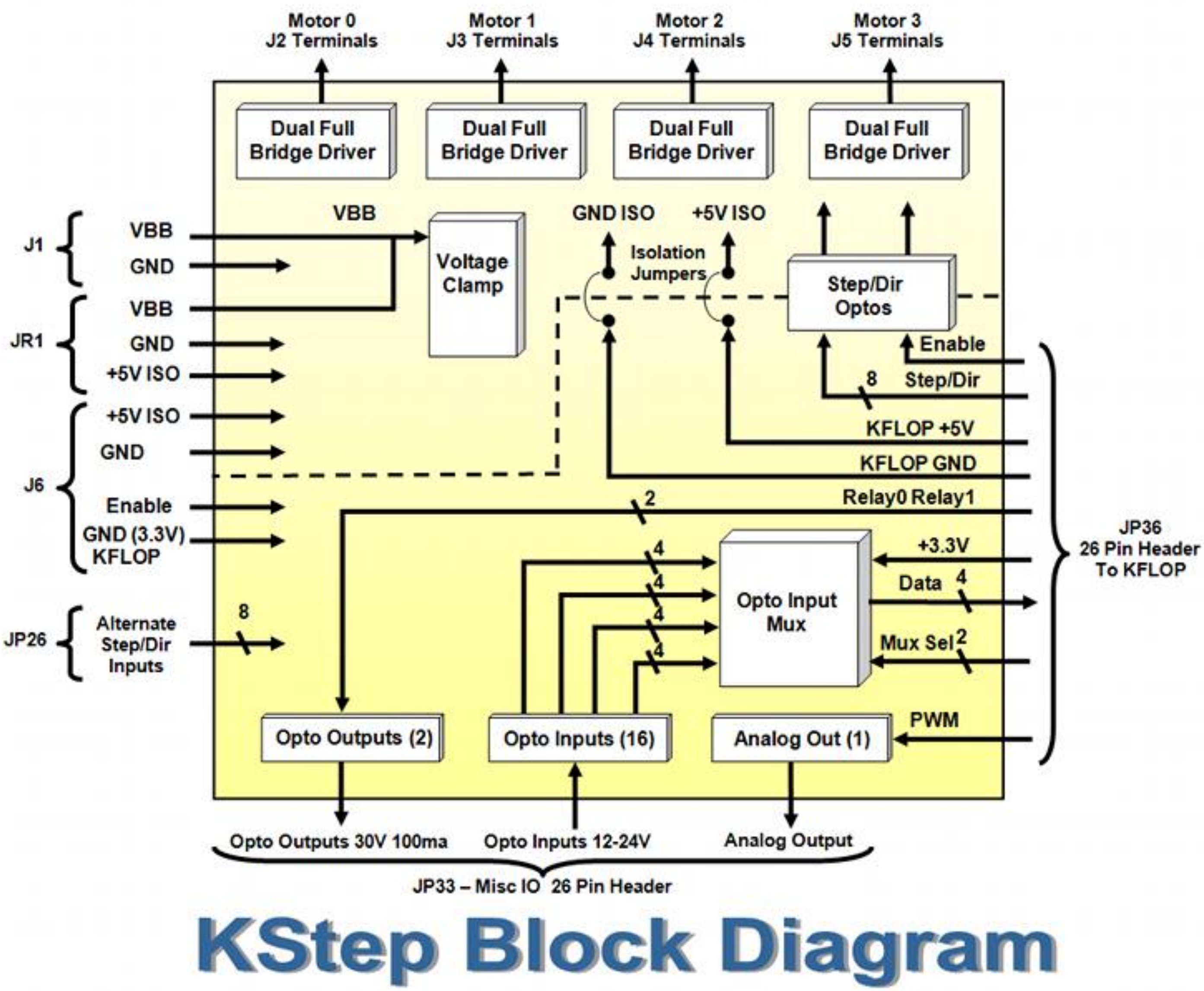
KStep allows all the Motor Power and +5V Mosfet circuitry to be totally optically isolated from the KFLOP/Kogna Step/Dir and Enable signals. To operate with total isolation remove JP34 and JP35. However in this mode an additional +5V supply must be provided and applied to either J6 or JR1. With the jumpers installed KStep will make use of KFLOP/Kogna +5V and KFLOP/Kogna GND coming in through connector JP36 and no additional supply will be required. The [Block Diagram] shows how the isolation is sectioned.

Regardless of these jumpers installed or not the 12-24V Optically isolated Inputs, the 2 Relay Driver outputs, and the Analog Output will always be fully optically isolated.

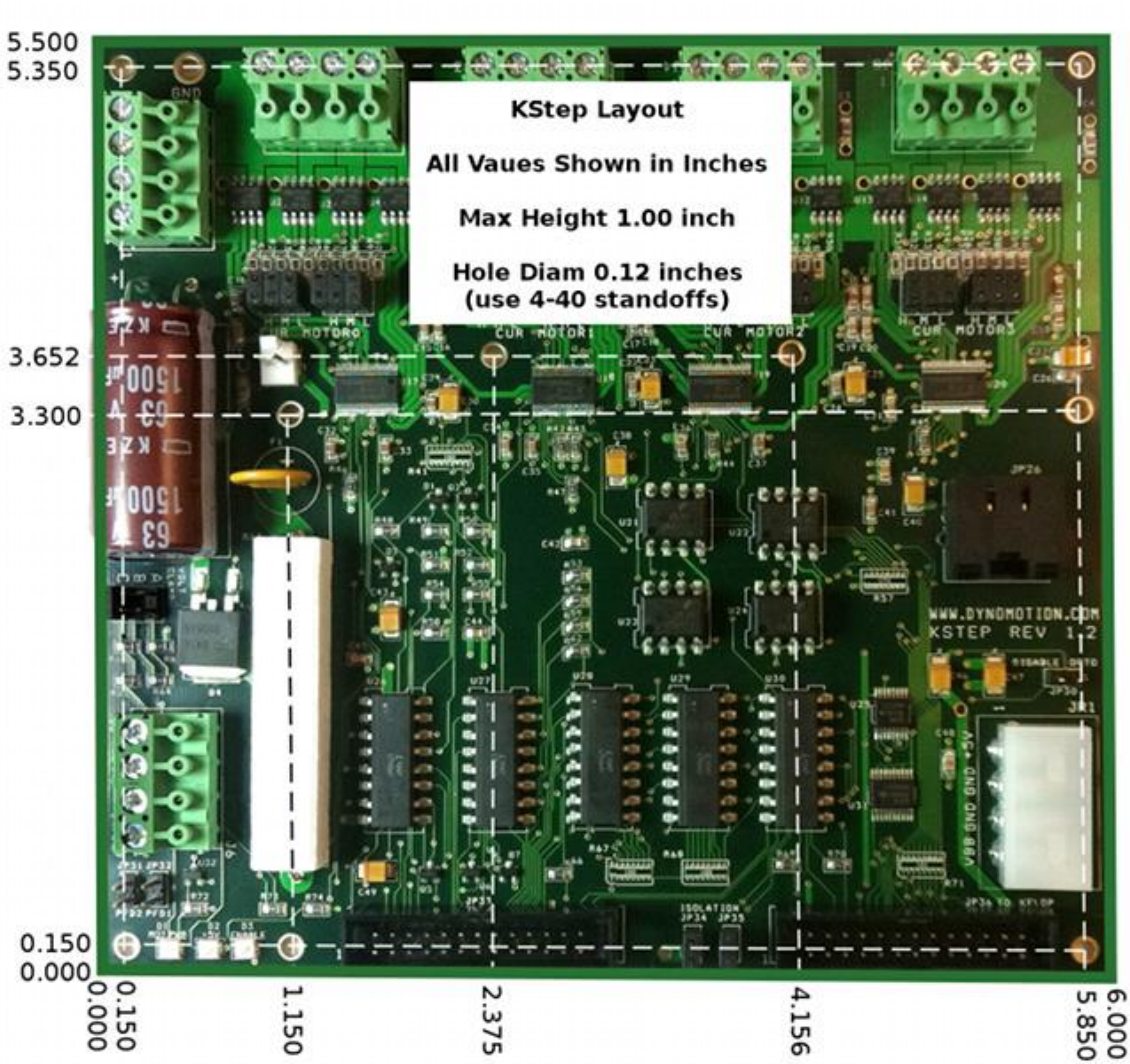
J6 - GND, +5V, Enable, 3.3V GND

J6 provides a connection point for +5V and GND to power the KStep Mosfet Drive Circuitry if operated in Isolated mode. The Enable connection allows a means of enabling KStep amplifiers if JP36 is not used to enable the amplifiers. Controller 3.3V GND is the same GND as on JP36 which is the KFLOP/Kogna or Controller Ground which is reference for the Step/Dir signals and Enable Signals. The Isolation Jumpers can be used to isolate or connect the two GND terminals on this connector.

KStep Block Diagram



Board Layout



Using KStep

KStep is a high efficiency 4-axis microstepping drive that can drive four motors with up to 5Amps @ 48V each. In addition to the 4 motor drives, KStep also provides additional I/O features:

- Sixteen (16) 12-24V tolerant optically isolated filtered inputs
- Two (2) optically isolated relay driver type outputs, each good for 0.1A @ 30V
- One (1) isolated PMW to Analog output

All controller signals are 3.3V/LVTTL compatible. An on-board voltage clamp circuit to protect against regenerative over voltage is also included. KStep is designed to snap together with KFLOP/Kogna for easy plug and play operation. A single 26-pin ribbon cable provides all the logic, power, Step/Dir Signals, Enable, Relay driver, Analog PWM, and Optically isolated input connections between the Kstep and KFLOP boards. If necessary, two KSteps can be connected to a single KFLOP/Kogna to drive a total of 8 motors.

Figure 1 - KStep

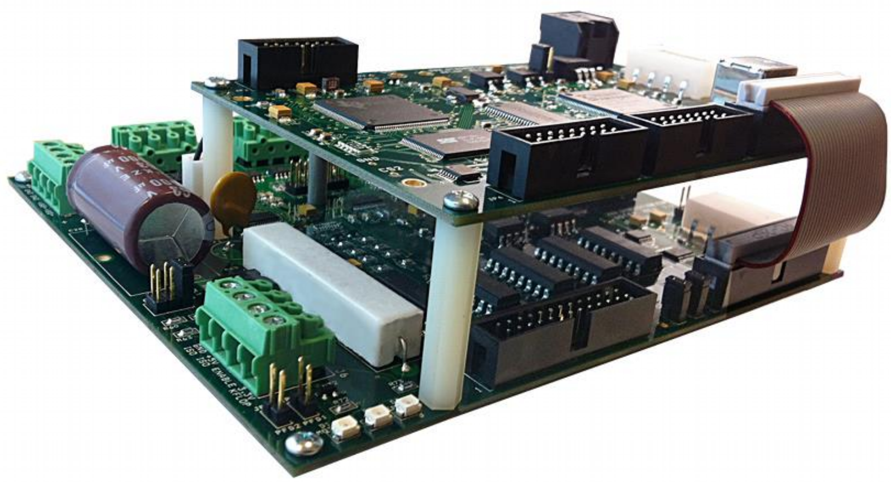


Figure 2 – KStep + KFLOP

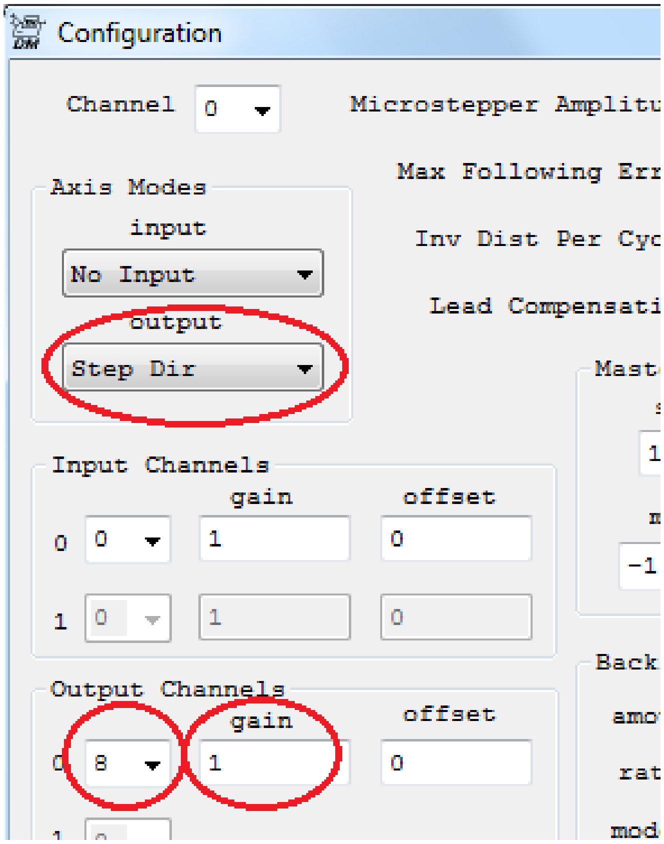
Configuring KFLOP/Kogna for use with KStep

Configuring KStep is fairly straightforward, as it is normally used in an open loop system (although it is possible to operate in closed loop stepper mode) so all feedback, servo, PID, Filter, Feedforward and commutation parameters are not used and can be ignored. KStep is effectively a Step/Dir driver, so it makes use of the Step/Dir Output mode in KFLOP/Kogna (See the items circled in red below - Note that output mode type "Step Dir" is selected). The screen shown is the Output Channel selection for KStep Motor channel #0. Note that output channel 8 is selected instead of 0 as you might assume. As KStep requires a LVTTL signal, rather than using Output channels 0 through 4, Output channels 8 through 11 respectively should be used instead, as they provide a LVTTL signal. See here for more information. An output gain of -1 may be specified to reverse the direction of motion.

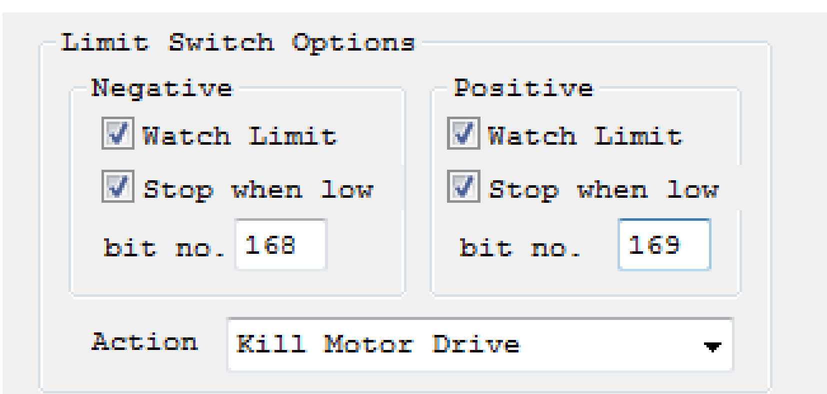
Note: for the pulse polarity and pulse time to be set properly, the following line of C Code needs to be executed after every power up. FPGA(STEP_PULSE_LENGTH_ADD) = 63 + 0x80; // set polarity and pulse length to 4us When using the default settings, there is marginal timing on direction setup which may result in "drift" caused by a microstep in the wrong direction when changing directions. This line of Code is already included in the Example C Initialization Programs (i.e. InitKStep3Axis.c)

FPGA(STEP_PULSE_LENGTH_ADD) = 63 + 0x80; // set polarity and pulse length to 4us

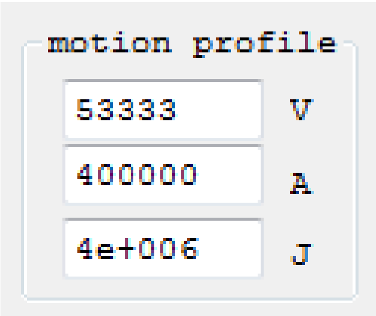
Example Axis configurations are also provided as KStepAxis0.mot, KStepAxis1.mot, KStepAxis2.mot, KStepAxis3.mot



Example Axis configurations are provided in the C:\KMotion431\KMotion\Motors directory (assuming a default install location). KStepAxis0.mot, KStepAxis1.mot, KStepAxis2.mot and KStepAxis3.mot are the most basic configurations. Limit switch options are also configured on this screen. KStep's optically isolated 12-24V Inputs are commonly used to connect Limit switches and are referenced as Input Bits 168 through 183 inclusive. If you are using NC (normally closed) type limit switches, this means that the inputs are normally high and become low when activated. In this case the "Stop when low" option should be selected. If using NO (normally open) type limit switches, this means the inputs are normally low, and become high when activated. In this case, the "Stop when low" option should be cleared. For more info see here.



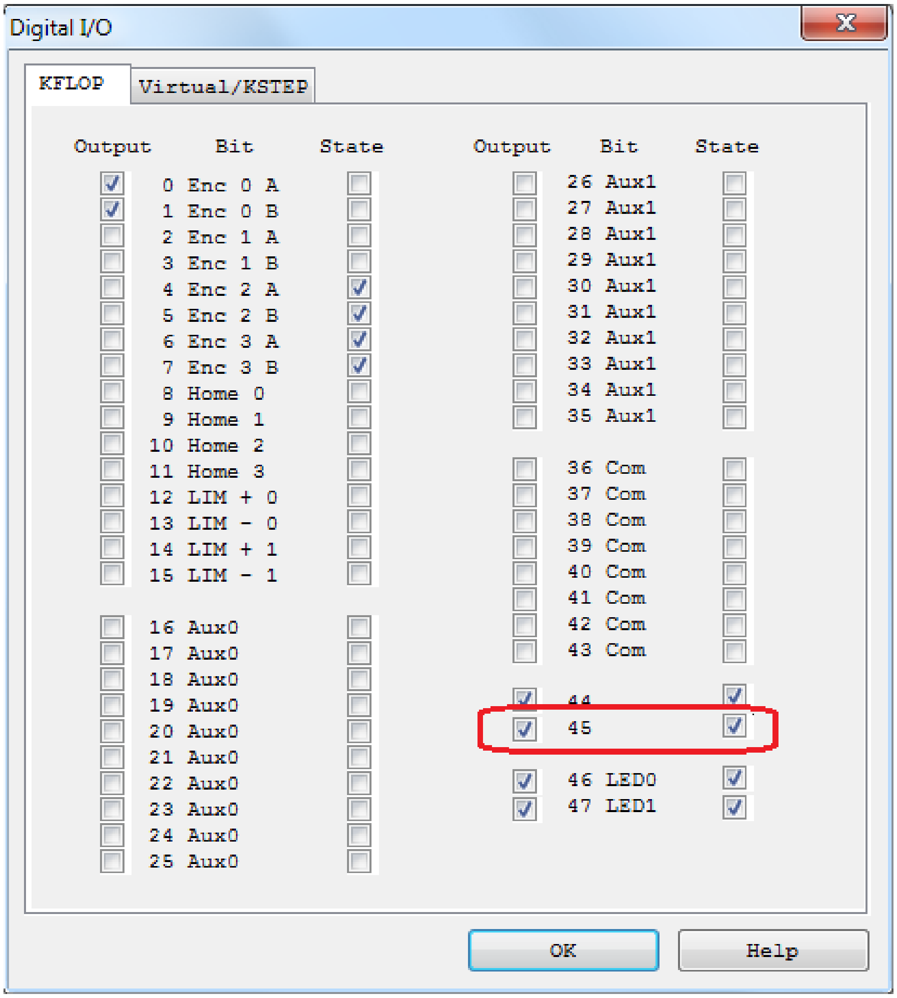
Motion profile settings on the Step Response Screen are used



Enable KStep Amplifiers

An amplifier enable signal is required to enable the KStep Full Bridge Amplifiers. One signal enables all four of the KStep amplifiers. The enable signal is positive true, meaning the drives are enabled when a 3.3V LVTTL (sourcing) signal is received. When using a 26-pin ribbon cable to connect KStep (JP 36) and KFLOP/Kogna (JP 7), this signal is routed to KFLOP's/Kogna's IO 45, which must be set as an output, and also set high to enable the drives. Note that the enable signal is only optically isolated from VBB and GND when KStep is in isolated mode.

If JP36 is not being used to connect KFLOP to KStep, then a 3.3V enable signal may be applied to a screw terminal on J6 instead. If two KSteps are being used (for driving 8 motors), then the screw terminals on J6 should be used as an output on the first KStep to daisy chain the enable signal to the second KStep.



For testing purposes KStep may be enabled using the Digital I/O Screen as shown below.

From C code the Amplifiers can be enabled with the following code.

```
SetBitDirection(45,1);   // set Enable Signal as Output
SetBit(45);              // Enable the KStep Amplifiers
```

It's possible to write code to enable the amplifiers as soon as motion is detected and disabled after a period of time with no motion. Here is an example:

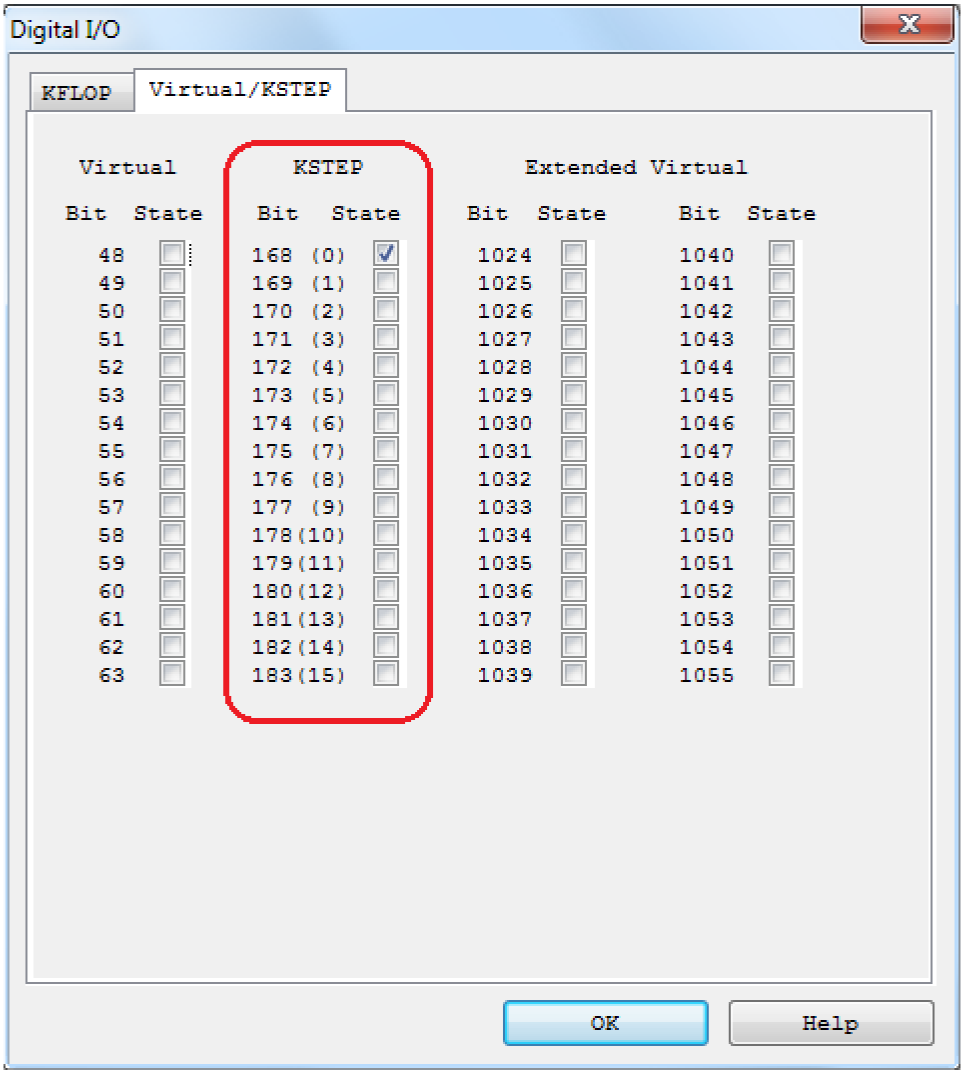
```
double T0, LastX=0, LastY=0, LastZ=0;
for (;;) // loop forever
 {
```

```
            WaitNextTimeSlice();


      // Service Amplifier disable after no activity for a while
      if (ch0->Dest != LastX || ch1->Dest != LastY || ch2->Dest != LastZ)
            {
      // we moved - enable KStep Amplifers
      SetBit(45);
                  T0 = Time_sec();  // record the time and position of
last motion
                  LastX=ch0->Dest;
                  LastY=ch1->Dest;
                  LastZ=ch2->Dest;
            }
            else
            {
                  if (Time_sec() > T0 + 10.0) ClearBit(45);
            }
      }
```

Digital Status

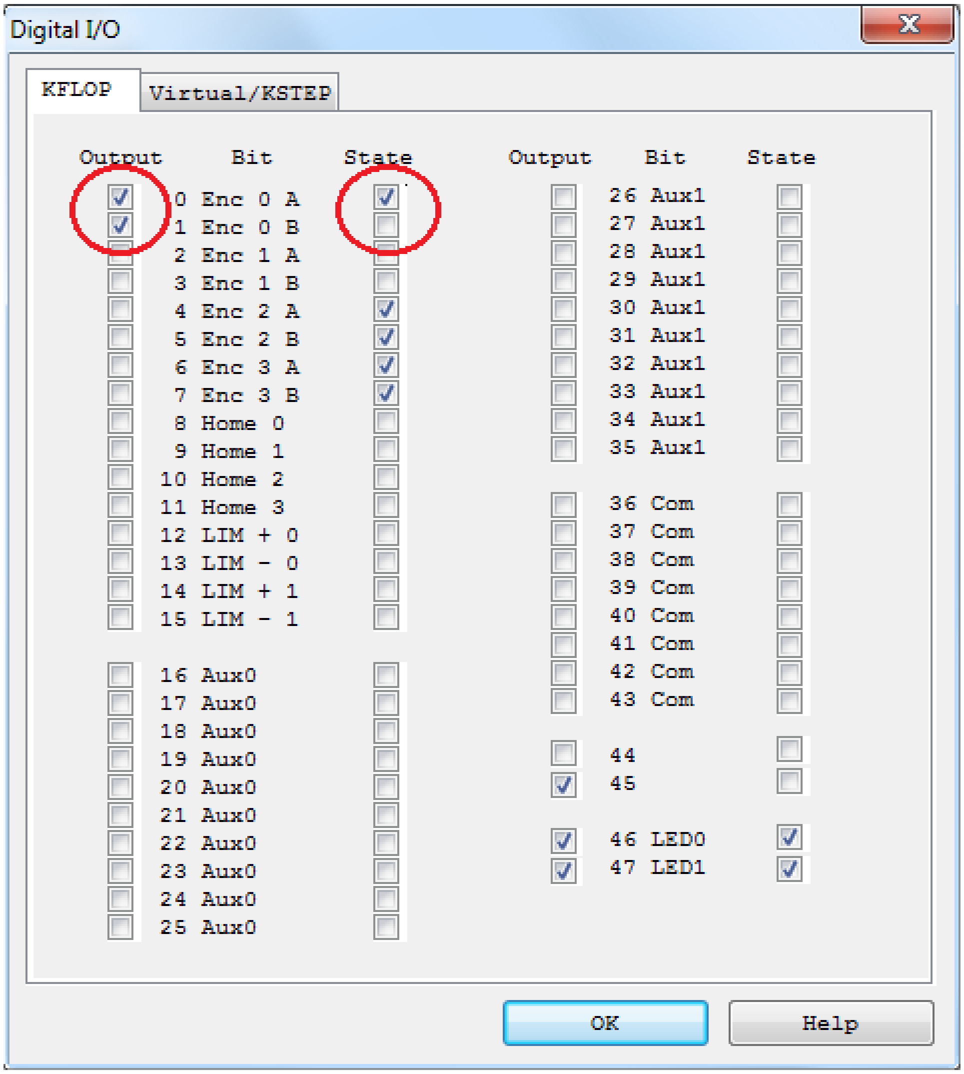
The state of the Optically Isolated 12-24V Digital Inputs can be observed by selecting the "Virtual/KStep" Tab on the Digital I/O Screen. Each input will be marked as active if current is flowing through the optical isolation circuit. For input circuits and pinouts see here. The KStep Opto Inputs are actually virtual inputs that are multiplexed in through KFLOP/Kogna I/O. KFLOP/Kogna can perform this multiplexing automatically by setting the global variable **KStepPresent.** The following line of C code should be added to your Init.c program:

KStepPresent=TRUE;



Relay Driver Outputs

The Optically Isolated 24V outputs can be observed and controlled by selecting the KFLOP/Kogna tab on the Digital I/O Screen. Each output will be marked as active if current is flowing through the optical isolation circuit. The KStep outputs are the same as the standard KFLOP/Kogna outputs except they are optically isolated and amplified to handle up to 24V. For output circuit and pinouts see.



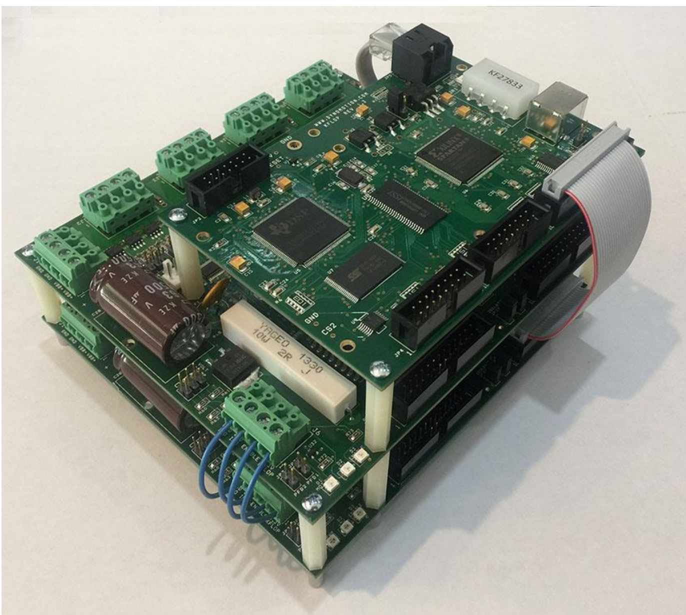
To make use of the outputs within a C program the following C code shows an example of enabling both the outputs, turning them on, then turning them off.

```
SetBitDirection(0,1);  //set as output
SetBitDirection(1,1);  //set as output

SetBit(0);  // turn output on
SetBit(1);  // turn output on

ClearBit(0);  // turn output off
ClearBit(1);  // turn output off
```

Connecting and Using 2 KSteps

Connect the RJ45 cable between KFLOP/Kogna JP5 and KStep#2 JP26. KFLOP's/Kogna's Step/Dir Generators 4-7 will then be connected to KStep#2. Configure as OutputChan0 12-15 to drive in TTL mode as required by KStep.

You will also need to provide power, GND, and Enable to KStep. The simplest method is usually to connect the 4 pins on J6 between the 2 KSteps.



Click here for more images.

Analog Output

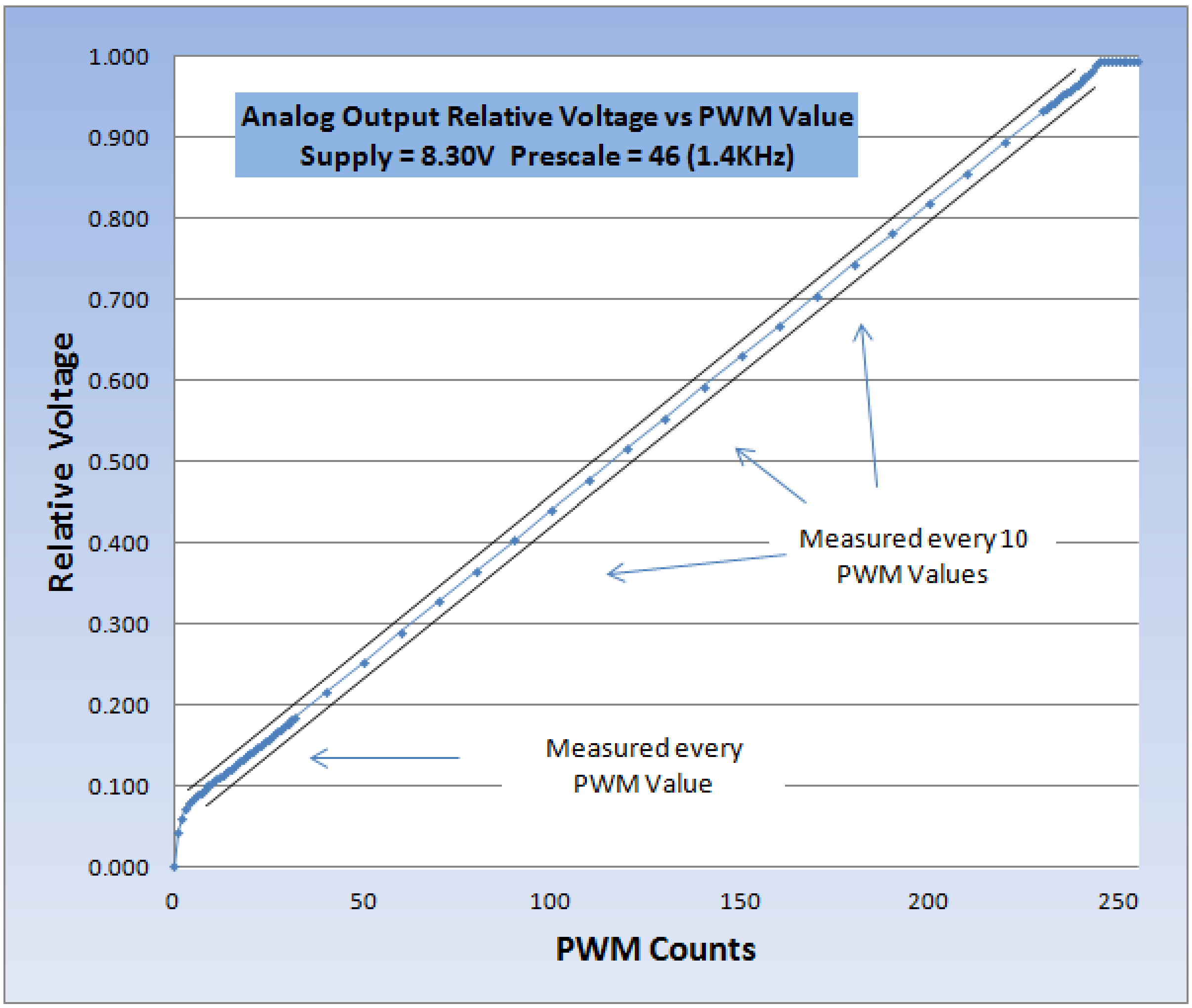
The KStep Analog output is a low Speed (10ms time constant) isolated analog output that is commonly used as a VFD Spindle Speed Control signal. KFLOP/Kogna outputs a 3.3V digital PWM on IO 44 to KStep which optically isolates and filters it to create an analog signal. Isolated power and ground (often supplied by the VFD) are required. The analog output voltage will be a proportion of the supplied voltage relative to the duty cycle of the PWM signal. For example a 75% high duty cycle will provide a voltage which is approximately 75% of the supplied voltage.

To configure KFLOP to output the appropriate PWM signal into KStep the configuration code below should be used. There is an FPGA option to move PWM0 (which normally is output on connector JP6 with the other 7 PWM outputs) over to JP7 Pin5 IO 44. IO 44 must be configured as an output; the PWM Prescale dividers set to a reasonable value (like?), and then enable PWM.

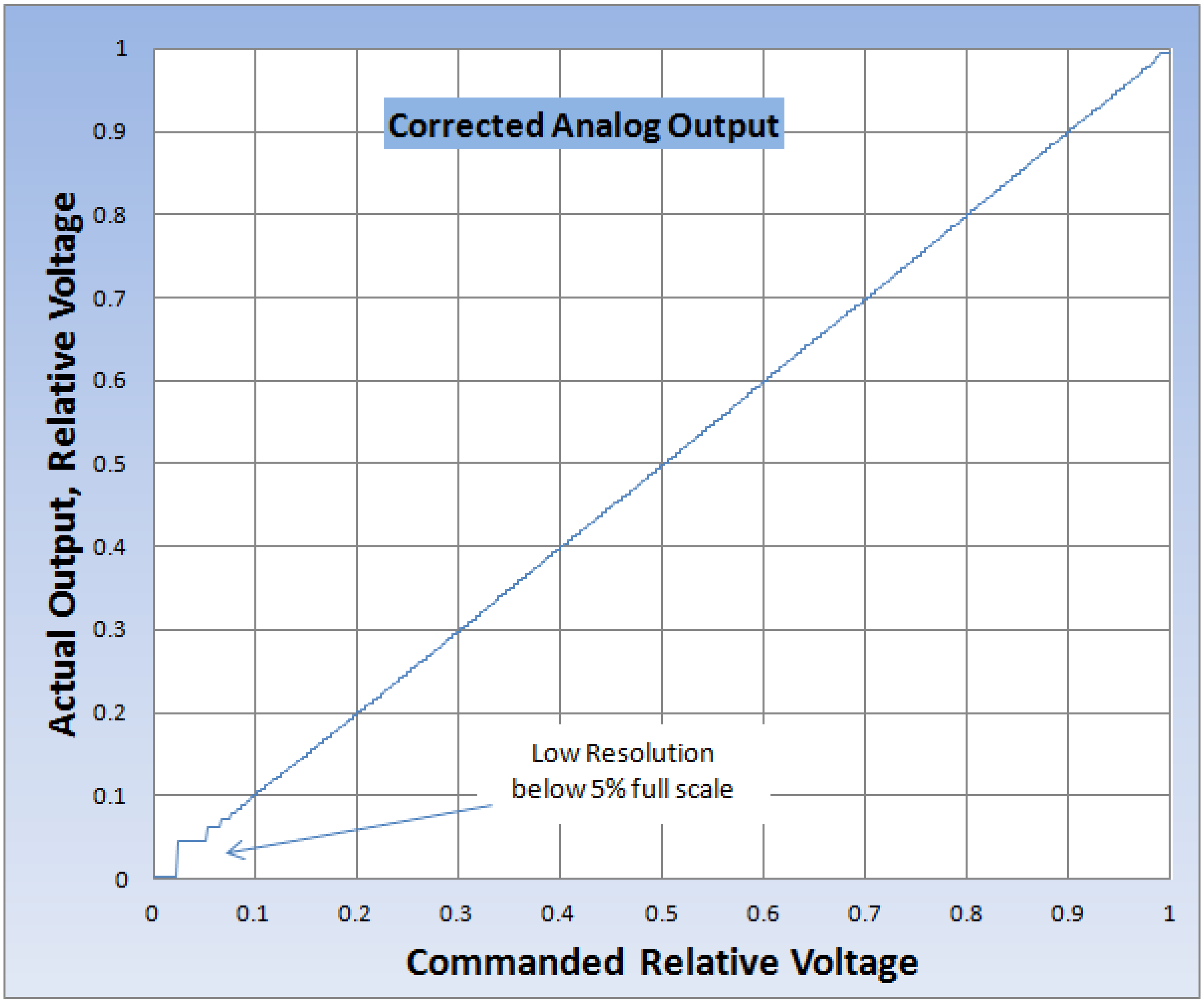
```
FPGA(KAN_TRIG_REG)=4;  // Mux PWM0 to JP7 Pin5 IO 44 for KStep

SetBitDirection(44,1);  // define bit as an output
```

```
FPGA(IO_PWMS_PRESCALE) = 46;          // divide clock by 46 (1.4 KHz)
FPGA(IO_PWMS+1) = 1;                  // Enable
```

The Analog output has nonlinearity near the 0% duty cycle region due to opto coupler pulse shaping effects. The nonlinearity can be mostly corrected in software through calibration. Below is a raw uncorrected analog output with a PWM frequency of 1.4KHz (PWM Prescale = 46).



Using a correction algorithm where the PWM value that will most closely produce the desired output voltage is determined from a table lookup approach for the first 7 PWM counts, and a simple linear interpolation is used for determining PWM settings 7 through 240 the linearized results are shown below. Although the signal is now linear, the resolution is low for settings below ~5% full scale.



Below is the C function used to linearize the output.

```
// PWM->Analog Correction
//
// assume very non-linear for first few count and linear thereafter
//
// Measure output ratio for first 0-7 counts then at 240

float V[]=
{
0.001,  // count = 0
0.031,  // count = 1
0.044,  // count = 2
0.054,  // count = 3
0.062,  // count = 4
0.069,  // count = 5
```

```
0.074,  // count = 6
0.079   // count = 7
};

float V240=0.970;

int CorrectAnalog(float v)
{
      int r;
      float v2=2.0f*v;
      // compare with half way points to determine closest count
      if (v2 < V[1]+V[0]) return 0;
      if (v2 < V[2]+V[1]) return 1;
      if (v2 < V[3]+V[2]) return 2;
      if (v2 < V[4]+V[3]) return 3;
      if (v2 < V[5]+V[4]) return 4;
      if (v2 < V[6]+V[5]) return 5;
      if (v2 < V[7]+V[6]) return 6;

      // must be 7 or higher do linear interpolation

      r = (int)(7.5 + (v-V[7])/(V240-V[7])*(240.0f-7.0f));
      if (r>255) r=255;
      return r;
}
```

KStep Basics Tutorial

Introduction

KStep is an add-on stepper driver for KFLOP and it is perhaps the easiest way available to start controlling stepper motors.

This guide will help you with the most basic KStep setup so that by the end of these instructions you will be able to control stepper motors using the KMotion setup/configuration screens, C-Code and G-Code. This guide is not comprehensive, however, and you should spend time reading the help pages for more in-depth information:

1. Main manuals page: http://dynomotion.com/Help/index.htm
2. KFLOP Connectors: http://dynomotion.com/Help/SchematicsKFLOP/ConnectorsKFLOP.htm
3. Kogna Connectors: https://www.dynomotion.com/Help/SchematicsKogna/ConnectorsKogna.html
4. KStep Connectors: http://dynomotion.com/Help/SchematicsKStep/ConnectorsKStep.htm
5. KStep Use and Settings: http://dynomotion.com/Help/SchematicsKStep/UsingKStep.htm
6. KStep Block Diagram: http://dynomotion.com/Help/BlockDiagramKStep.htm
7. KStep Specifications: http://dynomotion.com/Help/SpecificationKStep.htm
8. Dynomotion Forum (research and ask questions here): https://www.dynomotion.com/forum
9. CNC Forum (research and ask questions for CNC-specific applications here): http://www.cnczone.com/forums/dynomotion-kflop-kanalog/

KStep Tutorial Table of Contents:

What you need:

1. KFLOP/Kogna and KStep or KFLOP/ Kogna and 2xKStep combo with provided hardware

2. Dedicated 12V-48V Motor Power Supply to be wired into KStep as described below. Optional +5V power supply.
3. Stepper Motors
4. USB Cable (included with every KFLOP and KFLOP combo order) or Ethernet Cable (included with every Kogna and Kogna combo order)
5. Ohm Meter (or multi-meter)
6. General tools for electronics (wire cutters/strippers, small screwdrivers, etc.)

1. Software Setup

Start by ensuring you have the latest KMotion software installed on your computer. Download and install the latest KMotion software from This Software Download Page. Ensure the software installs and starts normally. If the software does not install properly, search the Dynomotion or the CNC Zone forums for help on common installation issues such as temporarily disabling antivirus software installation preventions, or enabling Windows unsigned software installations.

Note: Every screen in KMotion has a 'Help' button that will bring up a Help page that is focused on that particular screen.

2. Hardware Setup

2A. Providing +5V Board Power

In a static-free environment, remove your KFLOP/Kogna and KStep from the packaging and connect the two as shown here. The first step to powering up KFLOP/Kogna is to determine how to provide the +5V used to power the boards. Both KFLOP/Kogna and KStep require +5V for operation and there are a number of ways to provide it.

Note: Once +5V is supplied to the boards, it is available at every +5V terminal. Therefore do not supply +5V by more than one way or you will damage the boards and potentially your power supplies. The only exception to this is if KFLOP/Kogna and KStep are isolated by removing jumpers on KStep JP34 and JP35. With this isolation, +5V must be supplied to both KFLOP/Kogna and KStep as described below.

With jumpers installed on KStep's JP34 and JP35, the +5V and GND signals will be shared between both boards and only needs to be supplied in one way. The possibilities are:

1. On KFLOP/Kogna: Through the USB cable plugged into KFLOP/Kogna. The jumper on KFLOP's/Kogna's J3 must be installed for power to be received through the USB cable and the jumpers on KStep's JP34 and JP35 must be installed for KStep to receive the +5V signal. If board power is supplied this way, ensure +5V is not provided through KFLOP's/Kogna's JR1, KStep's JR1, or KStep's J6 connectors or the board and power supply could be damaged. Note that the USB cable should be of high quality and must meet USB 2.0 specifications for plug shell-to-shell resistance of <0.6 Ohms. When measuring resistance,

remember to subtract the resistance introduced by the measuring device by first measuring the probe-to-probe contact resistance.

2. On KFLOP/Kogna: +5V and GND through KFLOP's/Kogna's JR1 Molex® connector. The jumper on KFLOP's J3 must be removed to avoid receiving +5V from the USB cable in this case. The jumpers on KStep's JP34 and JP35 must be installed for KStep to receive the +5V signal - in this case do not provide +5V on KStep's JR1 or J6 connectors. Note that in this case, the +12V provided through the Molex connector is not used internally, but is routed to pins on the JP4, JP6, and JP7 connectors for convenience.

3. On KStep: +5V and GND through KStep's JR1 Molex connector. The jumpers on KFLOP's J3 must be removed and the jumpers on KStep's JP34 and JP35 must be installed in this case in order for KFLOP/Kogna to receive +5V power. Also note that if KStep's JR1 Molex connector is used, it is likely that +12V is being provided through this connector and therefore all VB connections on KStep will have 12V applied to them, and which would then be used for motor power. Therefore, do not provide motor power through KStep's J1 terminal.

4. On KStep: +5V and GND through KStep's J6 terminal block. The jumpers on KFLOP's J3 must be removed and the jumpers on KStep's JP34 and JP35 must be installed in this case in order for KFLOP to receive +5V power.

In the case where you want KFLOP/Kogna and KStep to have +5V supplied independently. Remove jumpers on KStep's JP34 and JP35 and then supply power through 1 or 2 and 3 or 4 above. Note that, in this case, the +5V power supplies should be different power supplies or the purpose of supplying board power independently is defeated.

Once the board power delivery method is determined, the boards are ready to power up. Provide +5V via your chosen method to power up the boards.

On power up, KFLOP/Kogna LEDs will illuminate as will the "+5V" LED on KStep indicating that both boards are receiving a +5V signal. Once motor supply power is fed into KStep as described below, the "MOT PWR" motor supply LED on KStep will illuminate.

2B. Check the firmware version and update if necessary

Next, ensure the software version on KFLOP /Kogna is up-to-date. In the KMotion software, open the Console screen and type the command "Version" into one of the command lines (if one of the command lines already contains the command there is no need to type it). Press the "Send" button next to the command to send the Version Command to KFLOP /Kogna to display the version of onboard firmware. If the firmware version does not match the version of KMotion you downloaded and are running (Help>About KMotion...), open the Config screen and press the "New Version" button. The pop-up window will automatically locate the necessary file for the sofware update, click "Open" and confirm the next pop-up. Allow a few seconds for the software to finish updating and after it is finished, cycle power to KFLOP /Kogna.

2C. Motor Power Supply Setup - Voltage Clamping

Voltage spikes due to Regenerative Braking and other sources can potentially raise the supply voltage up to a level that can damage electronics including KStep or the power supply. Voltage Spikes may also cause a power supply to trip a fault and shut down. KStep's Voltage clamp can be used to avoid these problems.

Motor power supply may be supplied through KStep's J1 terminal block OR KStep's JR1 Molex connector, but not both at the same time; and it is crucial that voltage clamps on KStep are set according to the motor power supply voltage level. KStep can handle motor power supplies from 12V to 48V and voltage clamp settings are designed to be 1V higher than common power supply voltages. It is critical that you clamp voltage on KStep at a value HIGHER than the motor power supply.

See this image for voltage clamp jumper location: KStep Connector Pinouts

Common power supply voltages are 12V, 24V, 36V, and 48v. KStep has a voltage clamping feature that can be set to 13V, 25V, 37V, or 49V volts. So whatever is the value of your motor power supply voltage, set the clamp to the next value higher than your motor power supply. Setting the voltage clamp to a value lower than the power supply will draw excessive current and is likely to cause damage to your KStep and/or the power supply.

Pin jumpers are provided with every KStep purchase. These are small plastic rectangular pieces with a metal coupling inside that connects two pins together when applied. You will use these to set the voltage clamp. Essentially, every jumper added to A, B, or C of the voltage clamping pins will reduce the voltage clamping by 12V. So, with no jumpers, voltage is clamped at 49V. With a jumper on A, voltage is clamped at 37V. With jumpers on A and B, voltage is clamped at 25V. With jumpers applied to A, B, and C, voltage is clamped at 13V (the lowest setting). Make sure voltage is clamped HIGHER than your power supply voltage or you will damage KStep.

Motor Power Supply Voltages	Required Number of Jumpers	Jumper Location	Clamping Level
37V to 48V	None	-	49V
25V to 36V	1	A	37V
13V to 24V	2	A+B	25V
12V Only	3	A+B+C	13V

2D. Motor Power Supply Setup - Connecting Motor Power to KStep

There are two ways to feed motor power into KStep, and they are incompatible with each other such that you must choose one or the other and not both or you will damage your power supplies and KStep.

The first and perhaps easiest way to connect motor power is through the Molex connector at position JR1 on KStep. Many power supplies have this standard connector and it plugs easily into KStep to provide motor power. **Caution:** Note that if JR1 is plugged into a power supply this way, all VB connections on the board will have that voltage active on them. Also, it is likely that +5V signals are provided by a power supply with this connector and so you need to make sure no other 5V signals are being provided as described in the Board Power section of this tutorial before plugging power into the board in this way.

The second way to supply motor power is through the screw terminals at position J1 on KStep. For applications requiring less than 10 Amps, simply screw in power supply leads into one of the respective VBB+ and GND terminals on J1. Do not supply power through JR1 if you supply power on J1 or you will damage KStep and your power supply.

Note: for applications requiring more than 10 Amps, both of the VBB and GND connections on J1 should be used.

Power Supply Voltage Selection is a fairly complex subject but as a rule-of-thumb the supply should be 10~15 times higher than the rated voltage of the steppers. Sometimes the voltage of the Stepper motor is not stated, in which case it can be determined by using Ohm's law (V = I x R) and the Current and Resistance specifications. For example, for a 3 Amp motor with a resistance of 0.6 ohms the motor would require 1.8V. A power supply of 18V to 27V might be used.

2E. Motor Current Settings

The current supplied to your motors must be configured to provide the correct level and KStep provides a physical method of doing this through the installation of jumpers on current setting pins on KStep.

Stepper motor current is generally specified on a per coil basis such as 670mA, or 0.67A, per coil. It is important to know what is the current per coil for your motors so you can set the correct current jumpers on KStep. The current setting pins are labeled on the board as "CUR MOTOR#" where # is the axis number from 0-3. KStep has a High, Medium, and Low current jumper for each motor coil (2 coils per axis). Set jumpers on the H (high), M (medium), and L (low) pins such that the value is the next level lower than your motor specifications. For example, if your motor has a per coil current rating of 670 mA, you will not set any jumpers for both coils - corresponding to a current setting of 0.63A. For a motor with a 3.5A current per coil specification you would set jumpers on H for both coils corresponding to a current setting of 3.13A.

Use the table below to determine how to set your jumpers according to the motor current ratings.

Motor Current per Coil Rating (Amps)	H	M	L	Current Setting (Amps)
0.63 to 1.24	Removed	Removed	Removed	0.63
1.25 to 1.87	Removed	Removed	Installed	1.25
1.88 to 2.49	Removed	Installed	Removed	1.88
2.50 to 3.13	Removed	Installed	Installed	2.50
3.13 to 3.74	Installed	Removed	Removed	3.13
3.75 to 4.37	Installed	Removed	Installed	3.75
4.38 to 4.99	Installed	Installed	Removed	4.38
5.00+	Installed	Installed	Installed	5.00

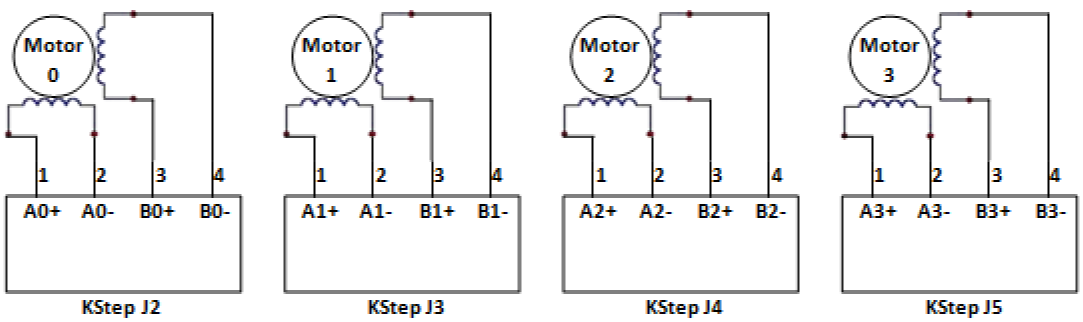
2F. Wiring Motors to KStep

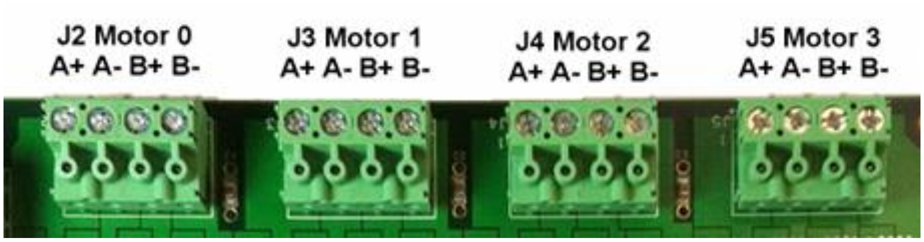
Stepper motors come in a variety of wiring configurations from 4 to 8 wires. KStep only needs 4 of these to drive a stepper since KStep does not require the center tap wires. KStep requires only the full bridge coils to drive the stepper and you can ignore center tap wires if your motor has them. For 8-wire steppers, you have a choice about how to wire them, in series or in parallel and would need to choose the power supply to fit the wiring method.

There are many resources on the web that describes Stepper motor wiring so we will not address the specifics here, but essentially you need only to wire in the coils across the "+" and "-" of the motor power output of KStep.

Warning: you can damage KStep if you wire the motors incorrectly. Make sure to measure the resistance between coils and that you have identified your coils properly before wiring into KStep. It is vitally important that each coil is wired into KStep properly.

Motors can be wired into connectors J2 through J5 on KStep according to the following diagram:





In addition, take note of the following default mapping as it is useful to know for the next steps (this mapping can be altered in software):

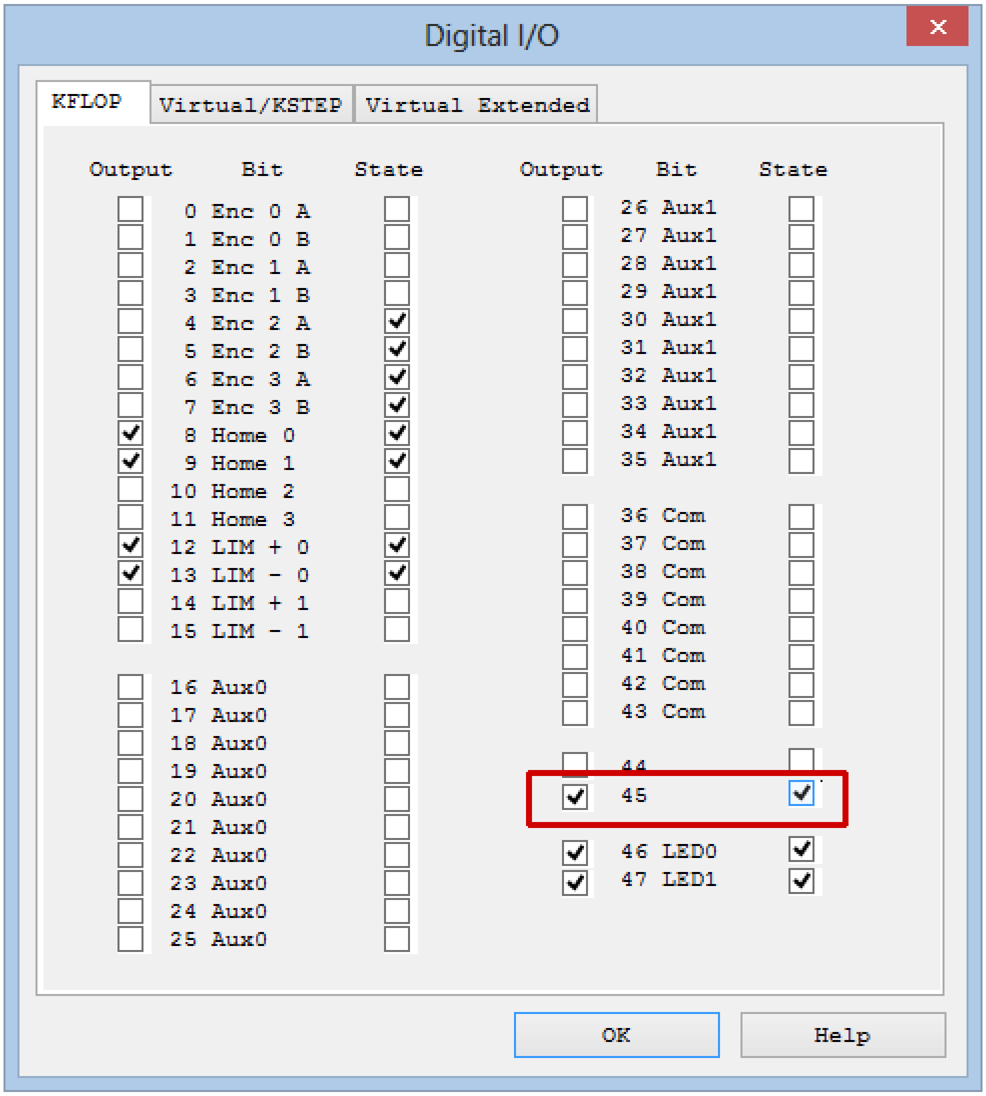
1. Axis Channel #0: Output Channel = 8 -> Motor Wiring J2
2. Axis Channel #1: Output Channel = 9 -> Motor Wiring J3
3. Axis Channel #2: Output Channel = 10 -> Motor Wiring J4
4. Axis Channel #3: Output Channel = 11 -> Motor Wiring J5

3. Driving Your Steppers

3A. Enabling Output

Once you have the voltage clamping, current limits and power supply set and wired, you are ready to begin moving your motors.

To do this, while the KStep LED D1 MOT PWR is illuminated, open KMotion.exe, select the Digital IO Screen, Set IO bit 45 as Output (enable the checkbox to the left of bit 45), Set IO 45 State to On (enable the checkbox to the right of bit 45). Verify that the KStep D3 ENABLE LED illuminates and that your motors hold position against a torque - in other words, you cannot move the motor shaft with your fingers. **In this state, the motors will become warm and may become too hot to touch.**



As a side note, this action can also be performed by entering the following commands in a C program or sending the commands using the Console screen:

```
SetBitDirection(45,1); //This sets the Enable Signal as Output
SetBit(45); //This Enables the amplifiers
```

Or by an external signal wired to KStep's J6 terminal.

3B. Setting Axis/Channel Parameters and Running an Initialization C Program (Init)

The goal in this section is to set the parameters that will ultimately go into the Init program. To do this, you will use two screens in KMotion: "Config & Flash" and "Step Response." If you plan to use an encoder for Closed-Loop Stepper Control, you would also use the "IIR Filters" screen as well, but it is not used for open-loop stepper control as described in this tutorial. You can view an explanation of how parameters are passed to/from KFLOP /Kogna, loaded, saved and copied to C Programs here: http://dynomotion.com/Help/FlashHelp/Parameters/index.html.

1. In KMotion, Open the "Config & Flash" Screen and select the "Channel" at the top of the screen for the axis you plan to use (0-3).

2. Since it is best to start with a known default, click on "Load Channel" and choose the corresponding "KStepAxis#.mot" file where # matches the chosen Channel. This will load the last saved settings in that file, which are set to default values that should work for a majority of stepper motors. In addition, loading this file puts the Channel into "Step Dir" Axis output mode, which is required for KStep.

3. After loading the Channel, open the "Step Response Screen" screen.

4. Notice at the top that the Channel number matches the Channel number on the Configuration screen. Note: changing the Channel on either the Configuration or the Step Response screen will change the Channel in the other screen.

5. The purpose of the Step Response screen in this tutorial is simply to help you set the Velocity, Acceleration and Jerk values to achieve smooth motor performance. These values are on the Step Response screen under "motion profile" labeled V, A and J, respectively.

6. Click on the 'Move' button. If Channel "0" is chosen, the motor wired to the J2 terminal should move a few degrees one direction and then back to the original position and, after a few seconds, a plot will appear in the Step Response Screen showing the Commanded Step movement. You can increase the step time (up to a maximum of 3.5 seconds) and size in this screen to get a larger movement.

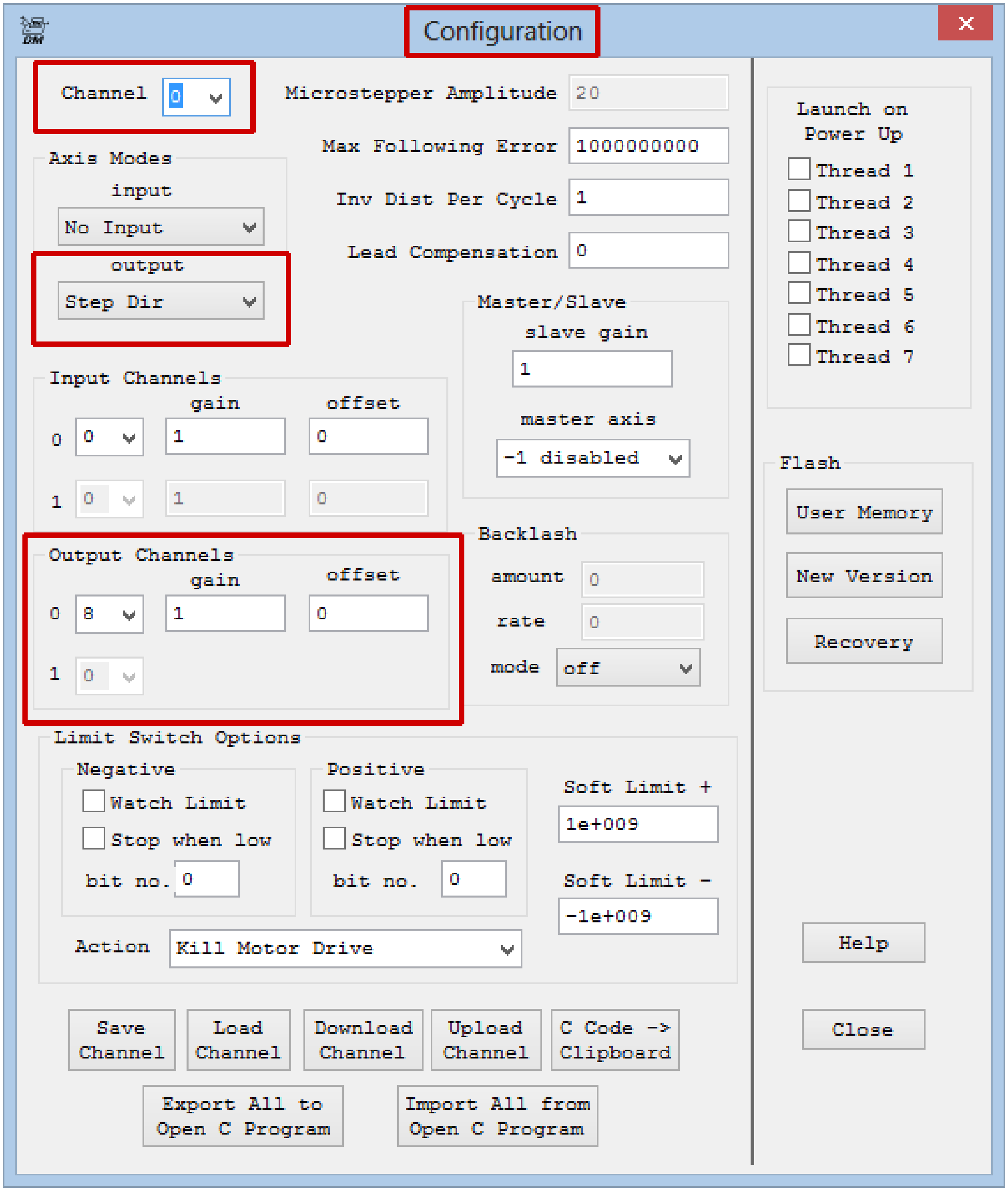
7. Now you can change the V, A, and J values until you are satisfied that the motor is moving smoothly - typically keeping A to be an order of magnitude or more than V, and J to be an order of magnitude more than A. If the motor stalls, seems jerky or misses steps, you can decrease the parameters until motion is smooth. In fact, it is recommended that you increase the parameters until the motor stalls, then decrease the parameters by 30-50%. Note: motors under load will perform differently so you may need to revisit this step once your motors are carrying weight. In that case, take precautions to ensure the system is not damaged if the motor experiences a stall. Click here for a discussion on choosing values for the Velocity, Acceleration and Jerk parameters.

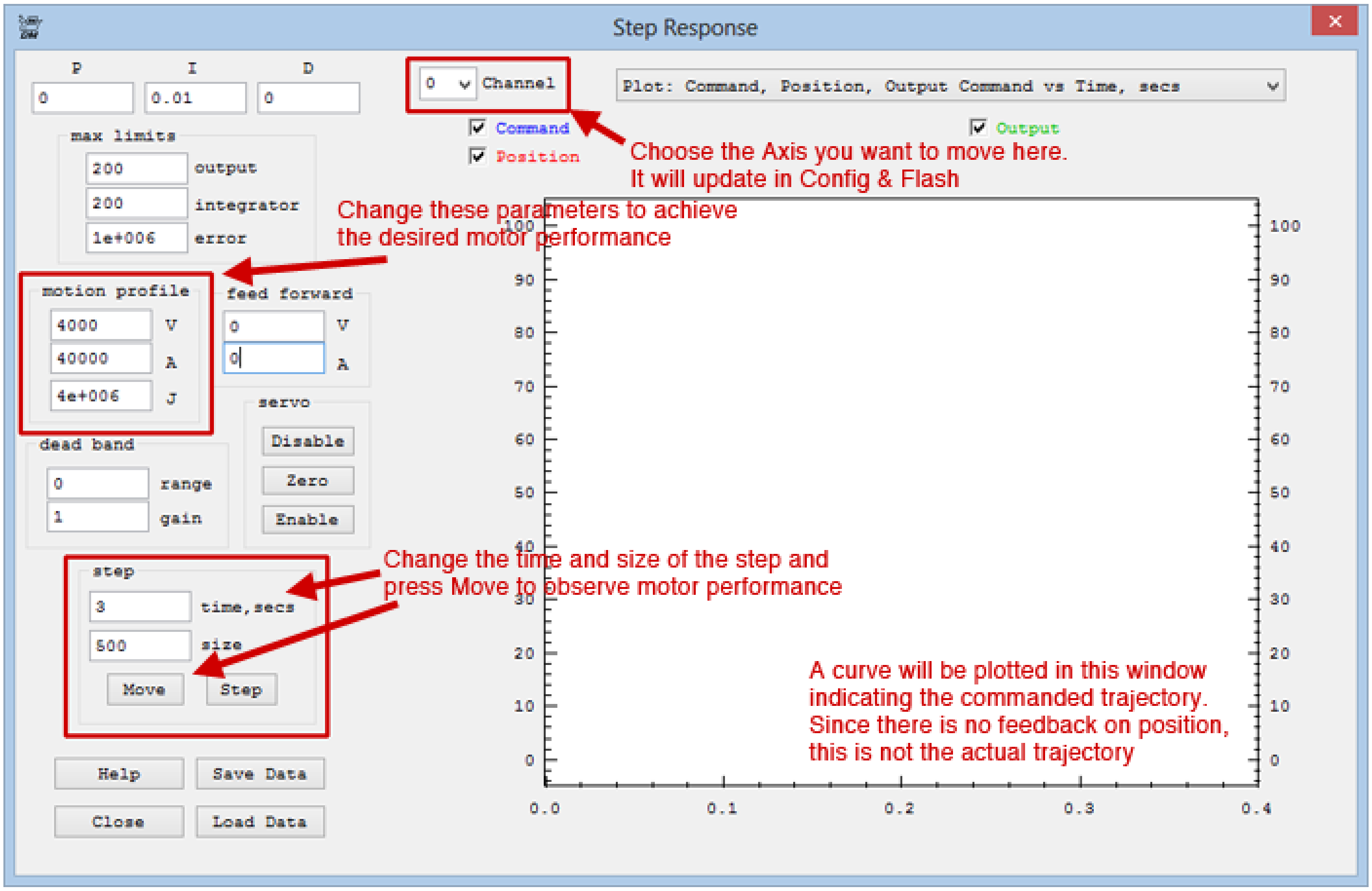
   The V, A, and J parameters are in terms of microsteps since KStep performs all movements using microstepping, which is 16 microsteps for every step. Therefore a Velocity value of 4,000 means 4,000 microsteps per second. You can relate this to your motor velocity by this equation:

Motor RPM = 60 * V / (16 * Step/Rev)

where V is the Velocity value on the Step Response screen and Step/Rev is the steps per revolution specification for your motor. For a motor with 200 steps/rev, a value of 4,000 for V equates to 75 RPM. Take note of the direction your motor spins and if you want it to spin in the opposite direction you can change the 'gain' value in the Configuration window to '-1'.

8. As the parameters are changed, they are automatically available to the Configuration window so they may be saved, downloaded to KFLOP /Kogna, copied to the clipboard or exported to an open C program. Furthermore, the parameters for each axis are persistent in that you can select a new Channel and the parameters for the previous Channel will not be lost unless the KMotion program is closed. Because of this, when you copy to the clipboard or export to an open C program, parameters for all axes are copied or exported to the C program (as long as the axis exists in the C program as shown below).


Configuration
Channel 0
Microstepper Amplitude 20
Max Following Error 1000000000
Inv Dist Per Cycle 1
Lead Compensation 0
Axis Modes
input
No Input
output
Step Dir
Master/Slave
slave gain
1
master axis
-1 disabled
Input Channels
gain
offset
Output Channels
Backlash
amount
rate
mode off
Launch on Power Up
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7
Flash
User Memory
New Version
Recovery
Limit Switch Options
Negative
Positive
Watch Limit
Stop when low
bit no.
Soft Limit +
1e+009
Soft Limit -
-1e+009
Action Kill Motor Drive
Help
Close
Save Channel
Load Channel
Download Channel
Upload Channel
C Code -> Clipboard
Export All to Open C Program
Import All from Open C Program



Once the parameters are set, you can save them to a file, export them to a C program or download them to KFLOP. To do this:

1. **Save to a file:** On the Config & Flash screen, click the Save Channel button and choose a file location and click Save. You can open this file later to reload those parameters.

2. **Download to KFLOP /Kogna:** On the Config & Flash screen, click the Download Channel button. This will load KFLOP /Kogna with the axis parameters.

3. **Copy to a C Program:** For this you must have a C Program open with the initialization parameters within **int main()**. An example C Program is located in the \KMotionXXX\C Programs\KStep folder and a good program to choose is the InitKStep3Axis.c, which has parameters for, and enables, 3 axes. Then simply click the 'Export All to Open C Program' and all available channels will be merged into the open C program if channel assignments are present there. This would then become an "Init" file.

4. **Copy to Clipboard:** This button simply copies the parameters to Window's Clipboard so you may paste into any other text window that you wish.

Here are the parameters for a single axis:

```
ch0->InputMode=NO_INPUT_MODE;

ch0->OutputMode=STEP_DIR_MODE;

ch0->Vel=115000;

ch0->Accel=50000;

ch0->Jerk=100000;

ch0->P=0;

ch0->I=0.01;

ch0->D=0;

ch0->FFAccel=0;

ch0->FFVel=0;

ch0->MaxI=200;

ch0->MaxErr=1e+006;

ch0->MaxOutput=200;

ch0->DeadBandGain=1;

ch0->DeadBandRange=0;

ch0->InputChan0=0;

ch0->InputChan1=0;

ch0->OutputChan0=8;

ch0->OutputChan1=0;

ch0->MasterAxis=-1;

ch0->LimitSwitchOptions=0x100;

ch0->LimitSwitchNegBit=0;

ch0->LimitSwitchPosBit=0;

ch0->SoftLimitPos=1e+030;
```

```
    ch0->SoftLimitNeg=-1e+030;
    ch0->InputGain0=1;
    ch0->InputGain1=1;
    ch0->InputOffset0=0;
    ch0->InputOffset1=0;
    ch0->OutputGain=1;
    ch0->OutputOffset=0;
    ch0->SlaveGain=1;
    ch0->BacklashMode=BACKLASH_OFF;
    ch0->BacklashAmount=320;
    ch0->BacklashRate=320;
    ch0->invDistPerCycle=1;
    ch0->Lead=0;
    ch0->MaxFollowingError=1000000000;
    ch0->StepperAmplitude=20;

    ch0->iir[0].B0=1;
    ch0->iir[0].B1=0;
    ch0->iir[0].B2=0;
    ch0->iir[0].A1=0;
    ch0->iir[0].A2=0;

    ch0->iir[1].B0=1;
    ch0->iir[1].B1=0;
    ch0->iir[1].B2=0;
    ch0->iir[1].A1=0;
    ch0->iir[1].A2=0;
```

```
    ch0->iir[2].B0=0.000769;

    ch0->iir[2].B1=0.001538;

    ch0->iir[2].B2=0.000769;

    ch0->iir[2].A1=1.92076;

    ch0->iir[2].A2=-0.923833;
```

Notice that the "ch0->" indicates it is setting parameters for Axis #0. Each axis will have all of these parameters and these are what are exported to the C program or copied to the clipboard when those options are selected in the Configuration screen.

Below is the example that shows 3 Axes being configured, enabled, and referenced into a coordinate system. This constitutes a full Init file to initialize a 3-axis system as discussed in the next section:

```
#include "KMotionDef.h"


// Defines axis 0, 1, 2 as simple step dir outputs

// enables them

// sets them as an xyz coordinate system for GCode


int main()

{

    ch0->InputMode=ENCODER_MODE;

    ch0->OutputMode=STEP_DIR_MODE;

    ch0->Vel=40000.000000;

    ch0->Accel=400000.000000;

    ch0->Jerk=4000000.000000;

    ch0->P=0.000000;

    ch0->I=0.010000;
```

```
ch0->D=0.000000;

ch0->FFAccel=0.000000;

ch0->FFVel=0.000000;

ch0->MaxI=200.000000;

ch0->MaxErr=1000000.000000;

ch0->MaxOutput=200.000000;

ch0->DeadBandGain=1.000000;

ch0->DeadBandRange=0.000000;

ch0->InputChan0=0;

ch0->InputChan1=0;

ch0->OutputChan0=0;

ch0->OutputChan1=0;

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=20.000000;


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=0.000769;
ch0->iir[2].B1=0.001538;
ch0->iir[2].B2=0.000769;
ch0->iir[2].A1=1.920810;
ch0->iir[2].A2=-0.923885;
EnableAxisDest(0,0);

ch1->InputMode=ENCODER_MODE;
ch1->OutputMode=STEP_DIR_MODE;
ch1->Vel=40000.000000;
ch1->Accel=400000.000000;
ch1->Jerk=4000000.000000;
ch1->P=0.000000;
ch1->I=0.010000;
ch1->D=0.000000;
ch1->FFAccel=0.000000;
ch1->FFVel=0.000000;
ch1->MaxI=200.000000;
ch1->MaxErr=1000000.000000;
ch1->MaxOutput=200.000000;
```

```
    ch1->DeadBandGain=1.000000;
    ch1->DeadBandRange=0.000000;
    ch1->InputChan0=1;
    ch1->InputChan1=0;
    ch1->OutputChan0=1;
    ch1->OutputChan1=0;
    ch1->LimitSwitchOptions=0x0;
    ch1->InputGain0=1.000000;
    ch1->InputGain1=1.000000;
    ch1->InputOffset0=0.000000;
    ch1->InputOffset1=0.000000;
    ch1->invDistPerCycle=1.000000;
    ch1->Lead=0.000000;
    ch1->MaxFollowingError=1000000000.000000;
    ch1->StepperAmplitude=20.000000;

    ch1->iir[0].B0=1.000000;
    ch1->iir[0].B1=0.000000;
    ch1->iir[0].B2=0.000000;
    ch1->iir[0].A1=0.000000;
    ch1->iir[0].A2=0.000000;

    ch1->iir[1].B0=1.000000;
    ch1->iir[1].B1=0.000000;
    ch1->iir[1].B2=0.000000;
    ch1->iir[1].A1=0.000000;
    ch1->iir[1].A2=0.000000;
```

```
ch1->iir[2].B0=0.000769;
ch1->iir[2].B1=0.001538;
ch1->iir[2].B2=0.000769;
ch1->iir[2].A1=1.920810;
ch1->iir[2].A2=-0.923885;
EnableAxisDest(1,0);

ch2->InputMode=ENCODER_MODE;
ch2->OutputMode=STEP_DIR_MODE;
ch2->Vel=40000.000000;
ch2->Accel=400000.000000;
ch2->Jerk=4000000.000000;
ch2->P=0.000000;
ch2->I=0.010000;
ch2->D=0.000000;
ch2->FFAccel=0.000000;
ch2->FFVel=0.000000;
ch2->MaxI=200.000000;
ch2->MaxErr=1000000.000000;
ch2->MaxOutput=200.000000;
ch2->DeadBandGain=1.000000;
ch2->DeadBandRange=0.000000;
ch2->InputChan0=2;
ch2->InputChan1=0;
ch2->OutputChan0=2;
ch2->OutputChan1=0;
```

```
ch2->LimitSwitchOptions=0x0;

ch2->InputGain0=1.000000;

ch2->InputGain1=1.000000;

ch2->InputOffset0=0.000000;

ch2->InputOffset1=0.000000;

ch2->invDistPerCycle=1.000000;

ch2->Lead=0.000000;

ch2->MaxFollowingError=1000000000.000000;

ch2->StepperAmplitude=20.000000;


ch2->iir[0].B0=1.000000;

ch2->iir[0].B1=0.000000;

ch2->iir[0].B2=0.000000;

ch2->iir[0].A1=0.000000;

ch2->iir[0].A2=0.000000;


ch2->iir[1].B0=1.000000;

ch2->iir[1].B1=0.000000;

ch2->iir[1].B2=0.000000;

ch2->iir[1].A1=0.000000;

ch2->iir[1].A2=0.000000;


ch2->iir[2].B0=0.000769;

ch2->iir[2].B1=0.001538;

ch2->iir[2].B2=0.000769;

ch2->iir[2].A1=1.920810;

ch2->iir[2].A2=-0.923885;
```

```
    EnableAxisDest(2,0);


    DefineCoordSystem(0,1,2,-1);


    return 0;
}
```

You can find more information on how to quickly import or export settings between the KMotion Screens and a C Program here.

3C. The Four Requirements for an Init C Program

At this point, you are ready to begin writing a C program to move your motors or simply use it as an Initilization file that allows KMotionCNC to control them. An initilization C program, which is also referred to simply as an Init file, will have four operations (all of which the InitKStep3Axis.c and InitKStep4Axis.c programs already have):

1. The following commands:

```
KStepPresent=True; //which tells KFLOP that KStep is connected
FPGA(KAN_TRIG_REG)=4; //Mux PWM0 to JP7 Pin5 IO 44 for KStep
FPGA(STEP_PULSE_LENGTH_ADD) = 63 + 0x80; //which sets polarity and pulse
length to 4us
```

2. Axis settings such as the velocity, acceleration and jerk parameters as described in the previous section along with the following command **for each axis**:

```
EnableAxisDest(#,0); //where # is the axis/channel number
```

3. Commands to enable all axes:

```
SetBitDirection(45,1); //This sets the Enable Signal as Output
SetBit(45); //This Enables the amplifiers
```

4. And then a command that defines which axes are in the coordinate system:

```
DefineCoordSystem(0,1,2,-1);
```

5. This command defines a coordinate system with 3 axes: 0, 1, 2 and the 4th axis is disabled by entering a '-1', otherwise a '3' would be used to include the 4th axis.

You can add more C code into this initialization file to control your system, or use Dynomotion's KMotionCNC program to run G-Code.

Since adding C code is more involved, that will be handled in a different tutorial, but here is an example of a simple infinite loop to move one motor that can be added into an init file:

```
for(;;) //loop forever
{

     MoveRelAtVel(0,500,100); //relative move Axis 0 for 500 steps at 100
     steps/second
     while (!CheckDone(0)) ; //ensure the move of Axis 0 is complete before
     moving on
     MoveRelAtVel(0,-500,100); //relative move Axis 0 for 500 steps in opposite

                                   // direction at 100 steps/second

     while (!CheckDone(0)) ; //ensure the move of Axis 0 is complete before
     moving on

} // close loop (loop still loops forever)
```

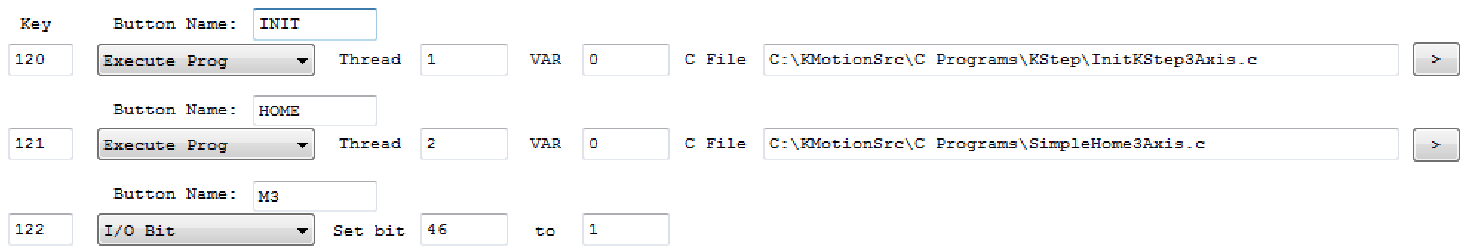
Then you only need to Save, Compile, Download and Run the program to have KFLOP execute the program. You can do this in steps on the C Program screen or all in one button click with the Save-Compile-Download-Run button on the top. To stop the program, press the "Stop" button on the main KMotion menu.

The next steps cover how to use KMotionCNC for basic movements.

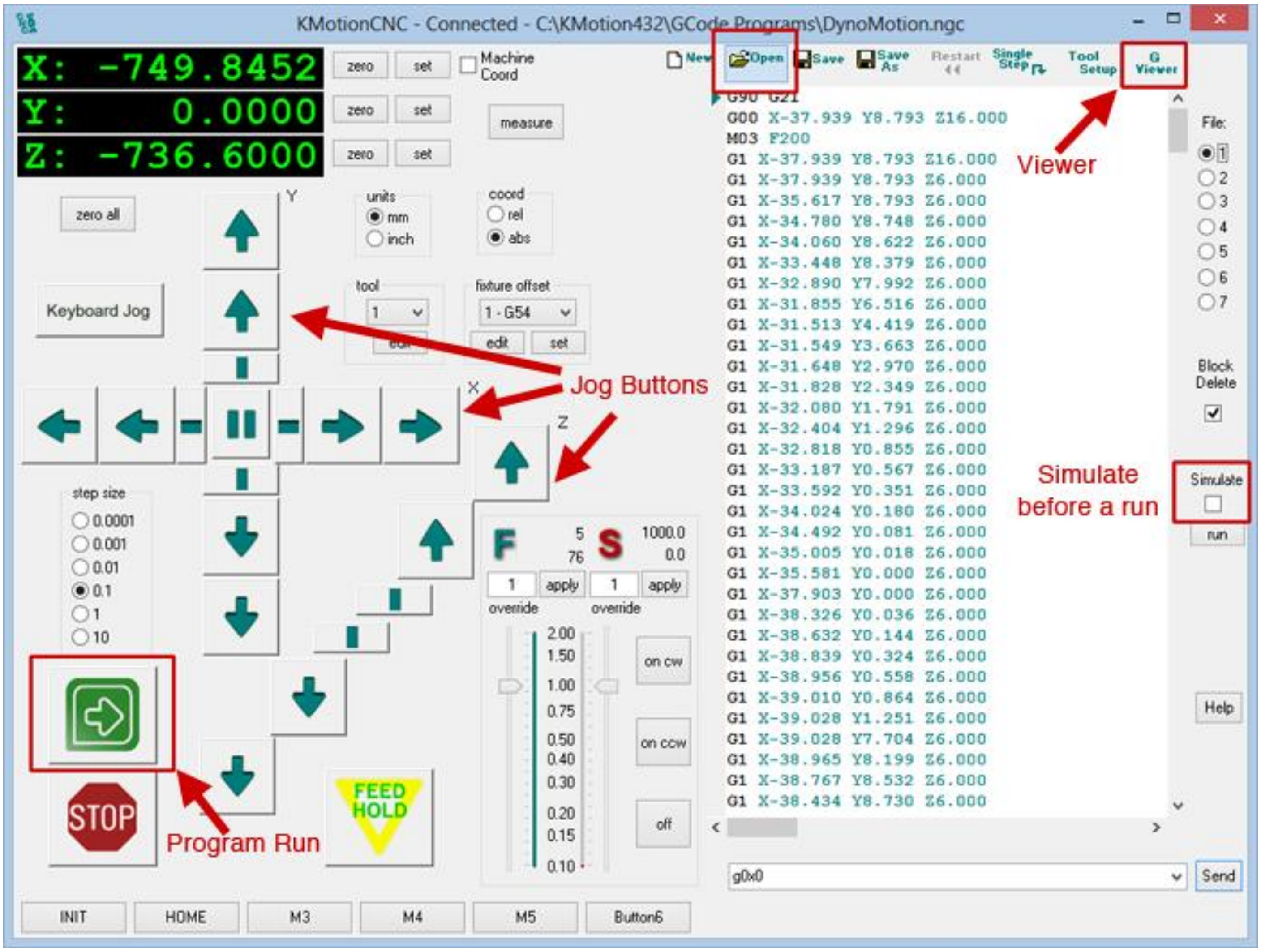
3D. Use KMotionCNC to jog your motors

After running an Init file such as InitKStep3Axis.c for initialization, you are ready to move your motors by using KMotionCNC. Try moving the motors by first clicking on the jog buttons (buttons with green arrows at left). If everything is initialized properly the motors should move when the respective jog button is clicked. These jog buttons are mapped to the axes by the "DefineCoordSystem(0,1,2,-1);" command discussed above. In this example, the X-axis is mapped to Axis #0, Y-axis is mapped to Axis #1, Z-axis is mapped to Axis #2 and the A-axis is not used. If pressing the jog buttons move the motors, then you are ready to begin running G-Code. To do this, within KMotionCNC choose "Open" and choose a G-Code program from the GCode Programs folder. If everything has been initialized correctly, the motors will move when the Start button is pressed to start the G-Code.

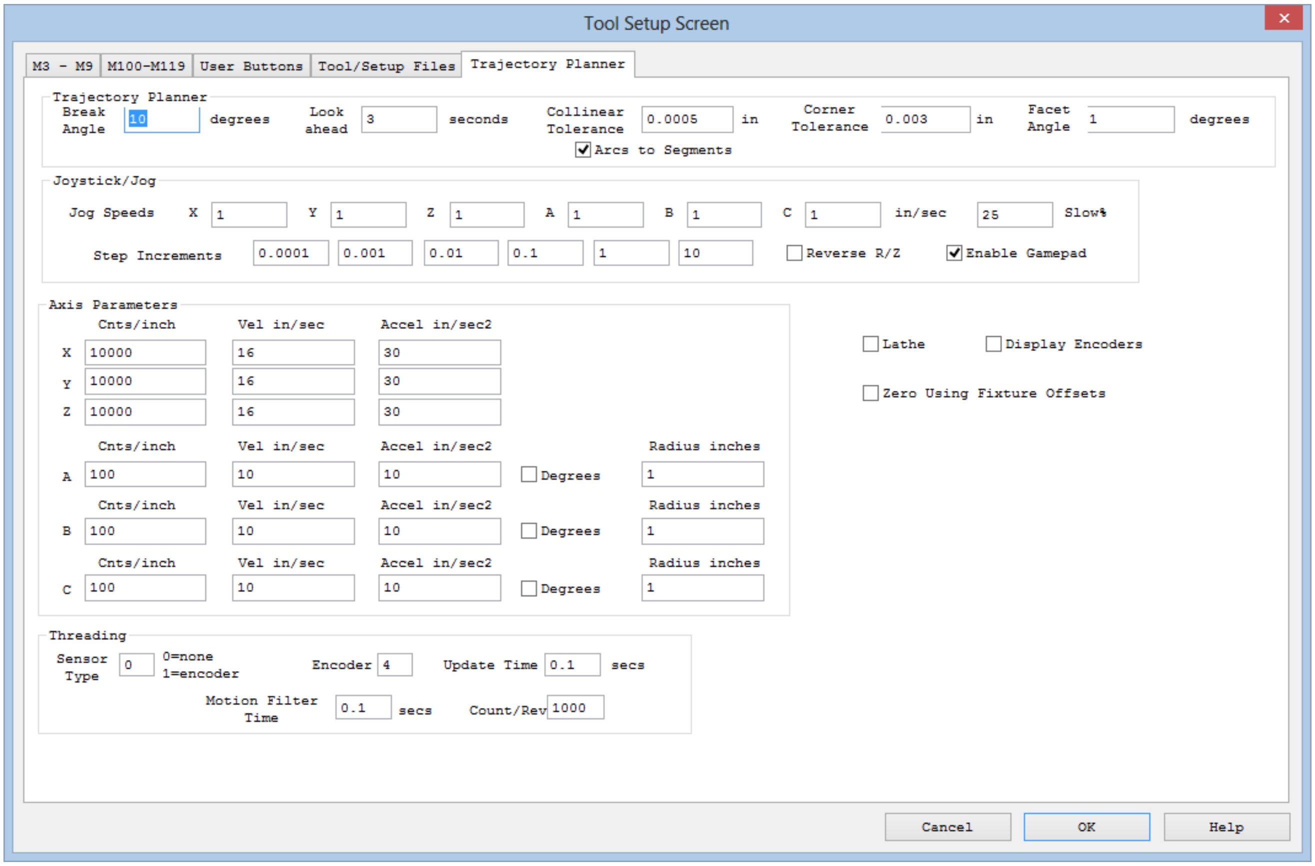
Note: An initialization C program such as the InitKStep3Axis.c must be run before operating with KMotionCNC will work. This initialization C program can be called with the 'Init' button in KMotionCNC once programmed to do so by setting the file location and Thread number for the INIT button within the 'Tool Setup' Screen on the 'User Buttons' tab in KMotionCNC:



The GUI of KMotionCNC can be configured based on how many axes you are using by changing the option for "Main Dialog Face" on the 'Tool/Setup Files' tab within the 'Tool Setup' screen. The screenshot below is for a 3-axis setup as you can tell from the display of only X, Y, and Z coordinates.



Note that the motors will not yet be moving accurately until the full parameters are set within KMotionCNC. To do this, you will set the Counts per inch (Cnts/inch) for each axis on the Trajectory Planner tab within the Tool Setup screen of KMotionCNC. You should also set the axis velocities and accelerations, and jog velocities and accelerations for your system on this tab.



You can measure the distance each axis moves for a given number of steps and then use this Cnts/inch value as a conversion factor so KMotionCNC knows how many steps to move for a given distance.

For greater accuracy, you can create a [Geocorrection File](). For a detailed discussion on Step and Direction Outputs, [see this page.]()

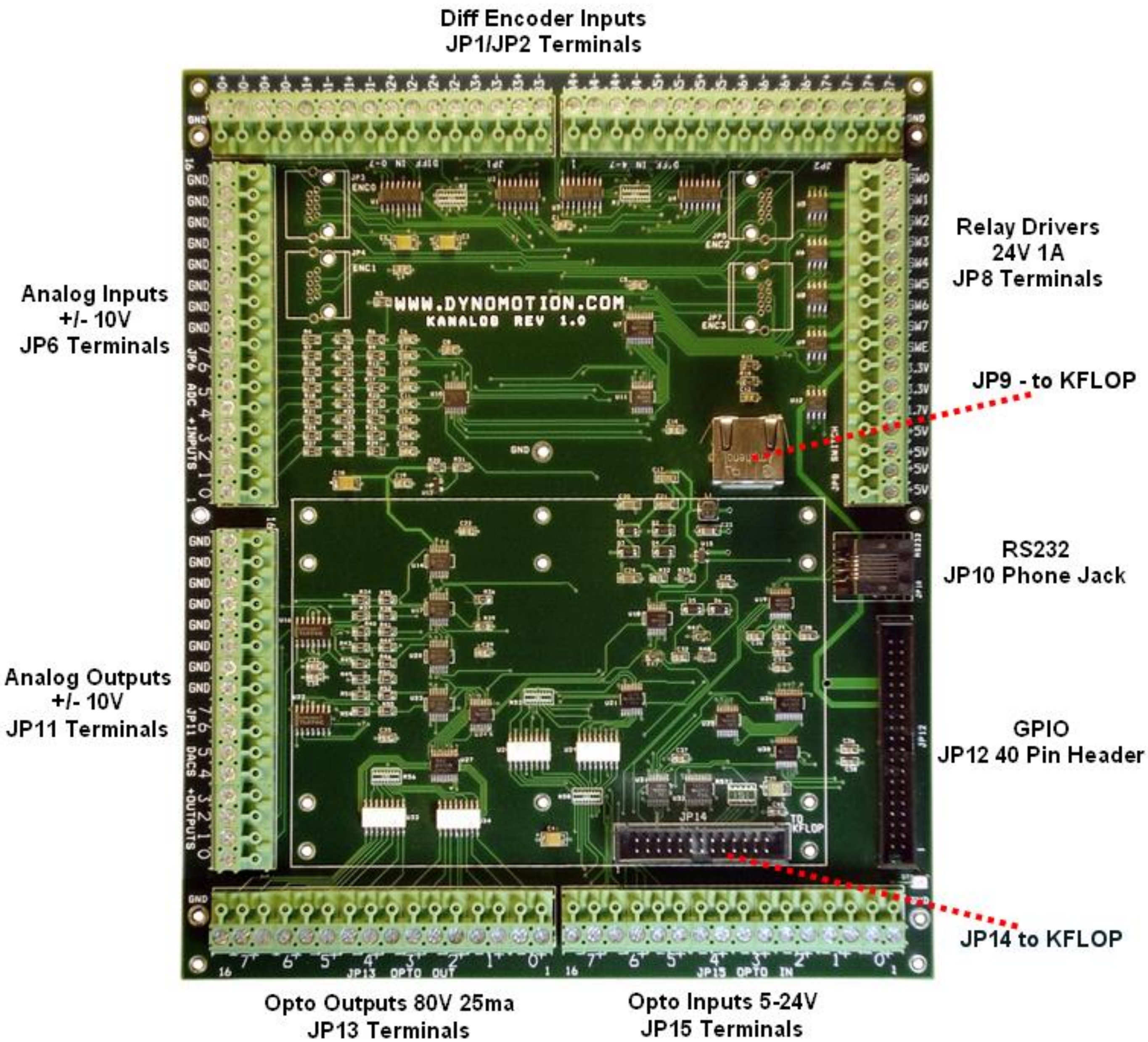
Once you understand everything in this tutorial, you will be ready to dive into the details of your KFLOP/KStep. See our [manual]() for more information.

Need to get up and running with KStep fast? View this high-level explanation of a basic KStep setup ([video]()):

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 Input Receivers	(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	0-40º C 0-40º 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

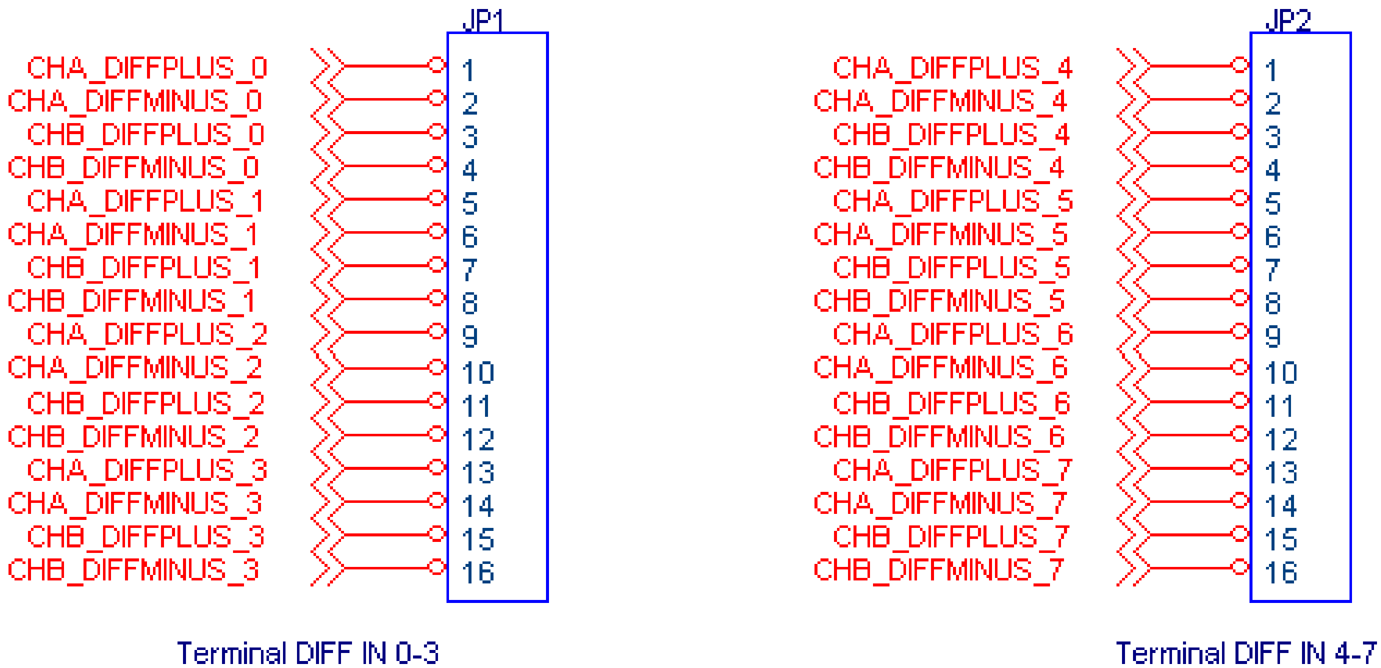
16 Differential Receiver Inputs are provided. Kanalog converts the differential signals to signal ended LVTTL signals and passes them through to existing KFLOP/Kogna I/O pins.

JP1 converts 8 of the signals and connects them to KFLOP/Kogna I/O bits 0-7 (KFLOP JP7 Pins 7-14) which are KFLOP's/Kogna's 4 encoder A/B input channels.

JP2 converts 8 of the signals and connects them to KFLOP/Kogna I/O bits 36-43 (KFLOP/Kogna 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

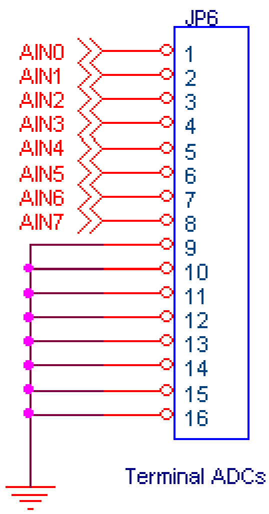
These differential inputs are moderately damped with an internal 470ohm resistor connected across the + to - inputs. If additional termination is required an external resistor may be connected.



JP6 - Analog Inputs +/- 10V

(8) +/- 10V analog inputs are provided. These are mapped and referenced as ADC inputs 0-7 for KFLOP/Kogna 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.

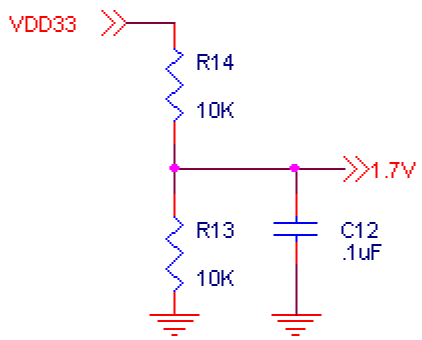
Caution! Inductive loads MUST have a reverse diode connected in parallel with the load to avoid a high voltage spark when the switch opens. Failure to allow a re-circulating current path on any inductive coil such as a relay, solenoid, or motor is likely to cause permanent damage.

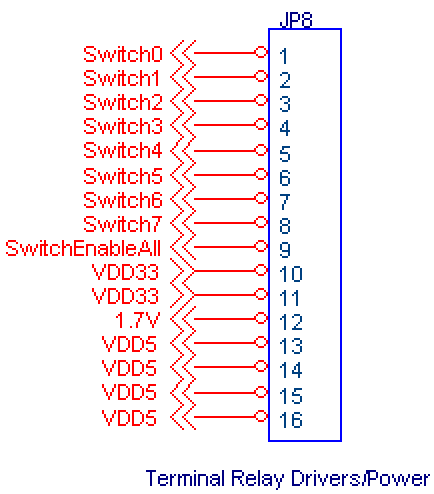
One Enable or Watchdog FET Switch Output is also available (24V @ 1Amp). This switch conducts after KFLOP boots, enables the +/-15V generator, and begins communicating with the board. It is recommended that this switch output is used an one of the conditions to enable main system power for motors and 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:

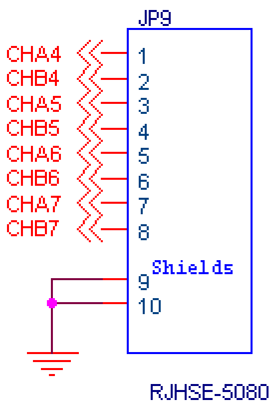
(4) +5V outputs are available to power low current (<100ma) external circuitry such as encoders. Normally +5V is applied to KFLOP/Kogna JR1 (4 pin white Molex connector) and passes through the ribbon connector to Kanalog. However it is also possible to feed +5V into the system via these terminals. If +5V power is fed into both KFLOP/Kogna and Kanalog the exact same +5V supply must be connected to both.





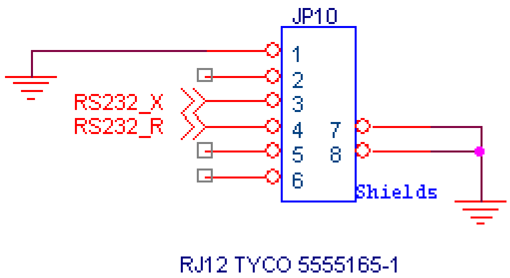
JP9 - Differential Signals 8-15 to KFLOP/Kogna

The second (8) of the 16 Differential signals pass through to KFLOP/Kogna through this connector. If only the first 8 differential are used then this connector is not required and the 8 KFLOP/Kogna inputs may be used for some other purpose.



JP10 - RS232

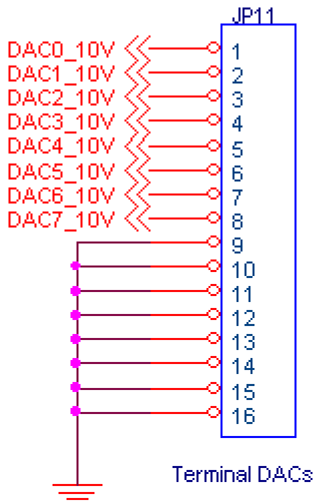
The JP10 6-pin phone connector provides 3-wire RS232 connectivity. JP10 Pin3 is used for transmitting data from Kanalog. JP10 Pin 4 is used for receiving data going into Kanalog. Received data is converted to LVTTL and routed to a KFLOP/Kogna I/O bit #44. Data from LVTTL KFLOP/Kogna I/O bit #45 is passed 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.



JP11 - Analog Ouputs +/- 10V

(8) +/- 10V analog ouputs are provided. These are mapped and referenced as DAC outputs 0-7 for KFLOP/Kogna 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.

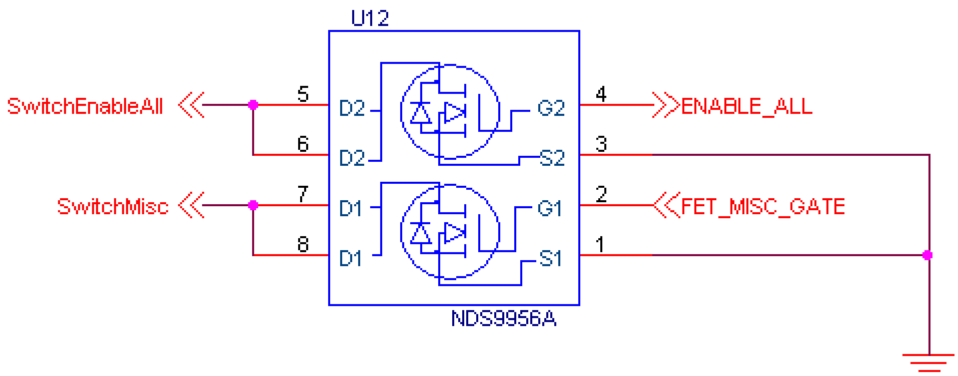
(8) 3.3V LVTTL Outputs are provided (OUT 0 - OUT_7) mapped as Kanalog Output bits #160-167.

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.

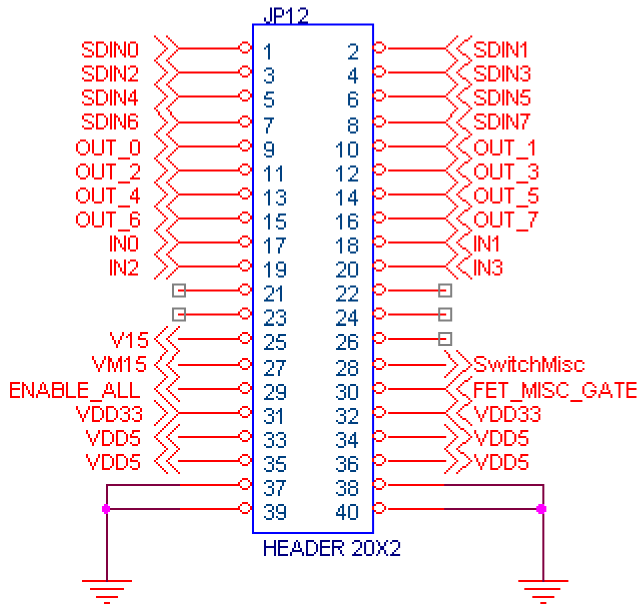
+/- 15V from the internal DC-DC generator is available on pins V15 (+15V) and VM15 (-15V). 70ma is available for external use for each supply.

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.



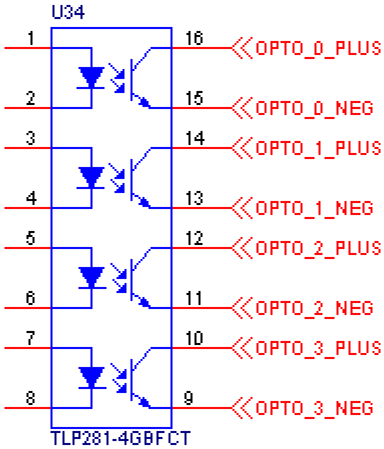
+3.3V and +5V are available on this connector. +3.3V is available as an output only (regulated down from the +5V). +5V is common to all other +5V signals and may be used as input or output.

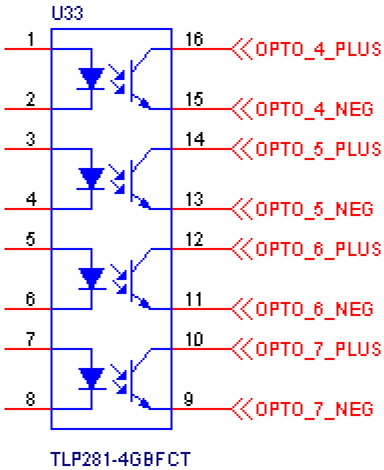


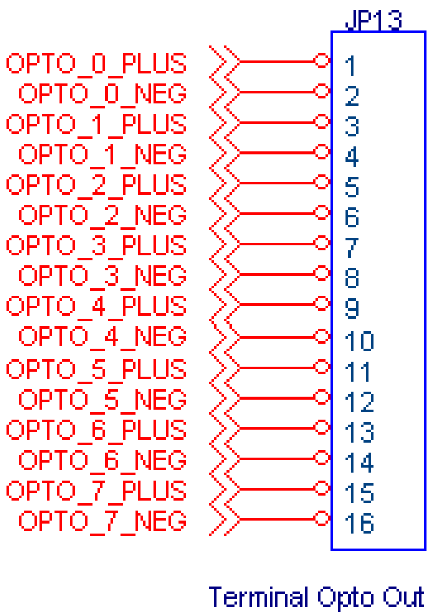
JP13 - Opto Outputs

(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.


U34
OPTO_0_PLUS
OPTO_0_NEG
OPTO_1_PLUS
OPTO_1_NEG
OPTO_2_PLUS
OPTO_2_NEG
OPTO_3_PLUS
OPTO_3_NEG
TLP281-4GBFCT


U33
OPTO_4_PLUS
OPTO_4_NEG
OPTO_5_PLUS
OPTO_5_NEG
OPTO_6_PLUS
OPTO_6_NEG
OPTO_7_PLUS
OPTO_7_NEG
TLP281-4GBFCT

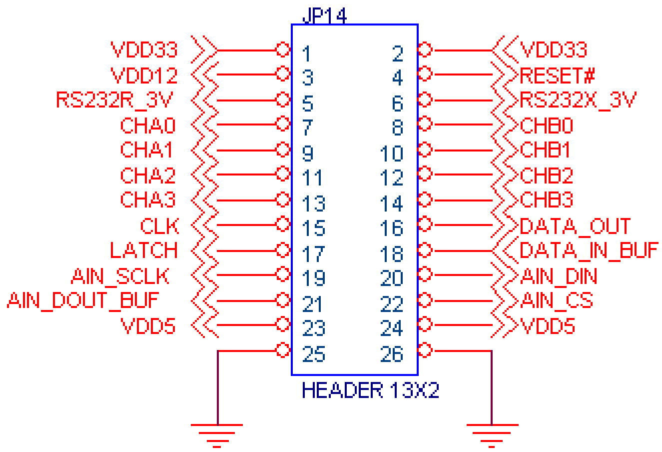

JP13
OPTO_0_PLUS
OPTO_0_NEG
OPTO_1_PLUS
OPTO_1_NEG
OPTO_2_PLUS
OPTO_2_NEG
OPTO_3_PLUS
OPTO_3_NEG
OPTO_4_PLUS
OPTO_4_NEG
OPTO_5_PLUS
OPTO_5_NEG
OPTO_6_PLUS
OPTO_6_NEG
OPTO_7_PLUS
OPTO_7_NEG
Terminal Opto Out

JP14 – KFLOP/Kogna

This is the main connection between KFLOP/Kogna 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/Kogna in parallel form.

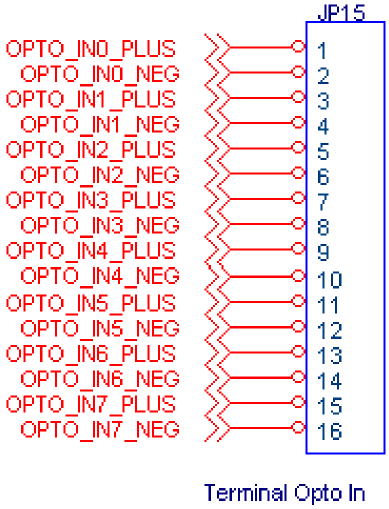
None of these signals should be used by the User.



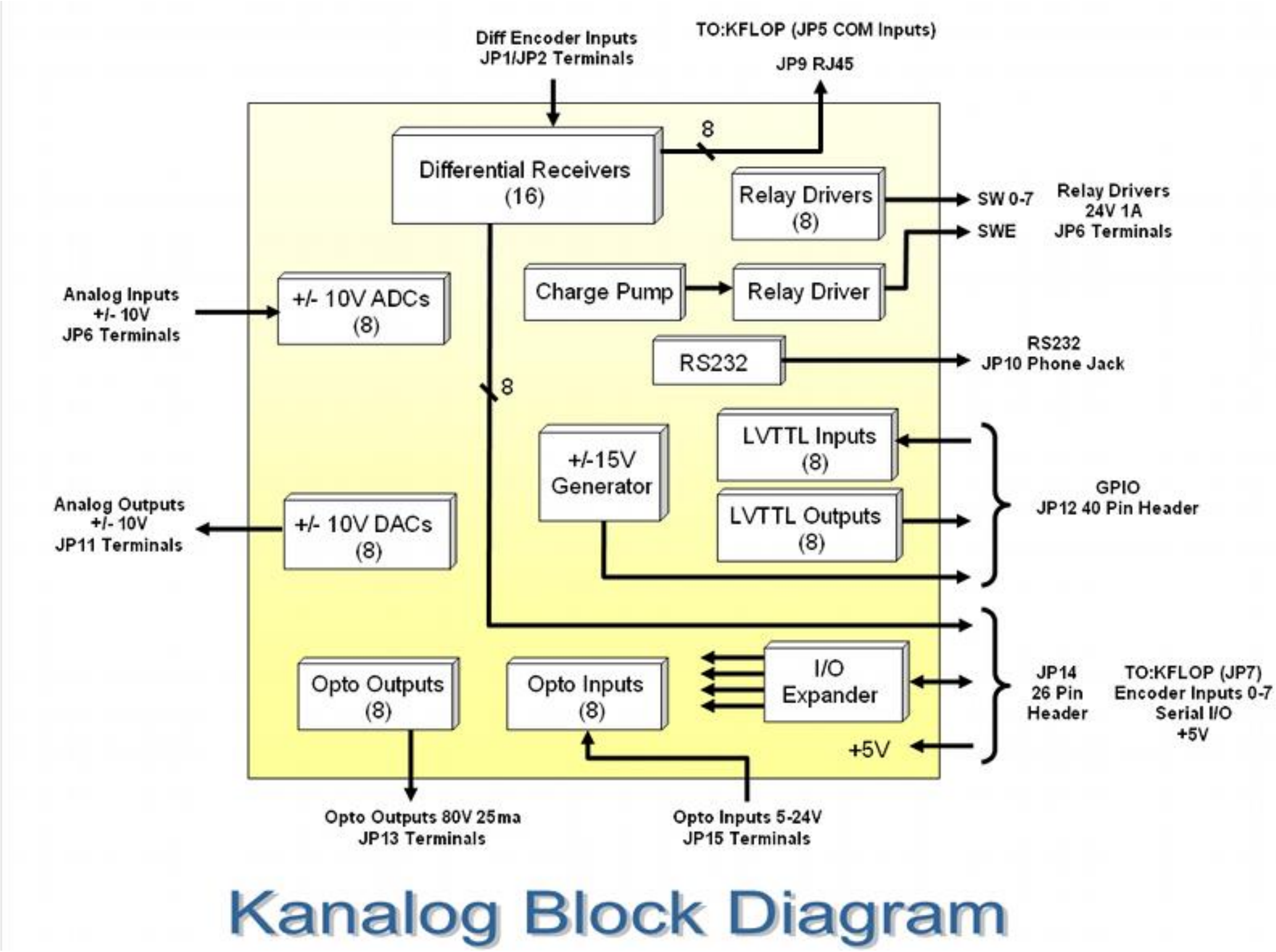
JP15 - Opto Intputs

(8) totally isolated and independent optically isolated inputs are provided.

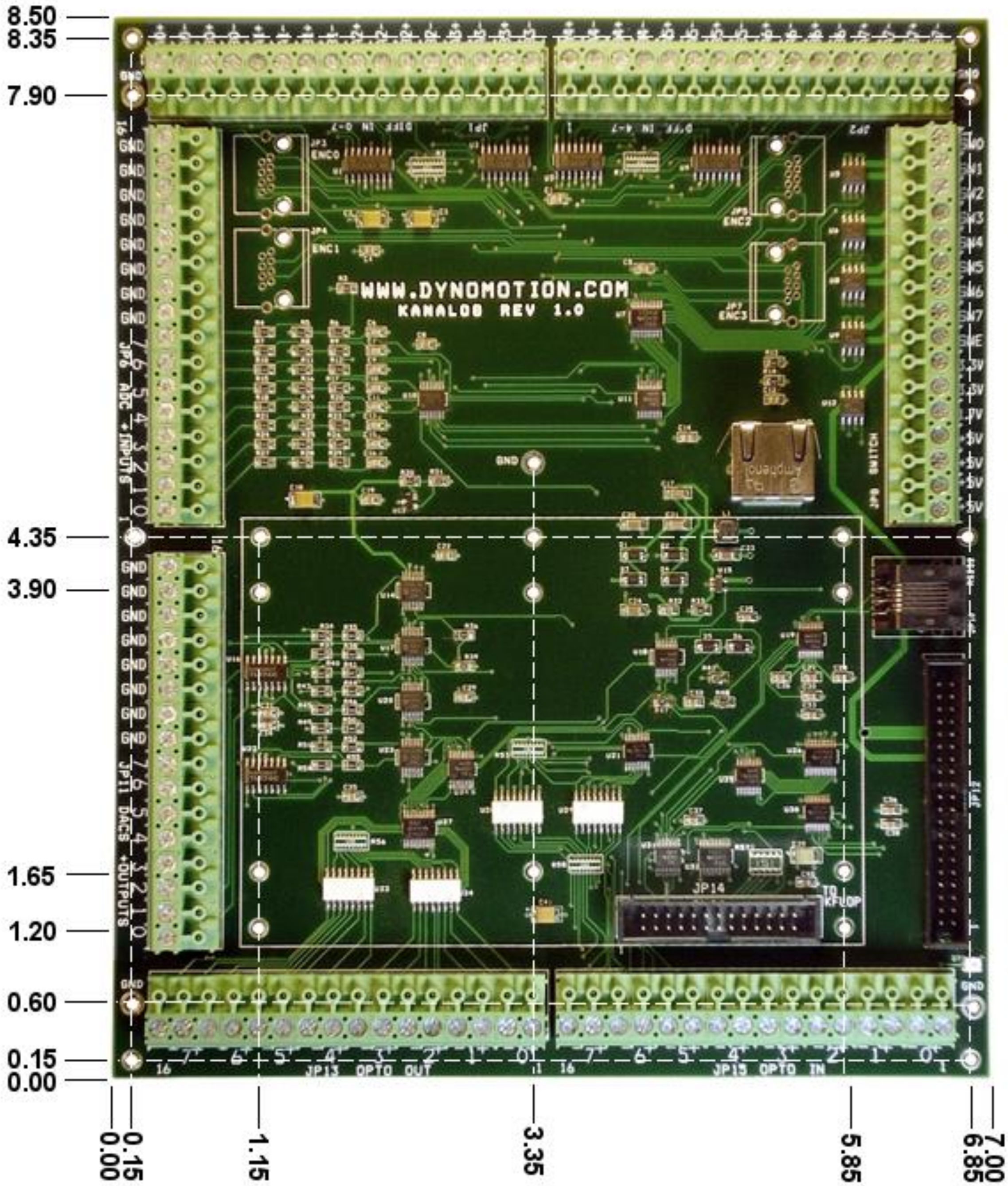
Input LEDs have a series resistance of 10Kohms and may be driven directly by any voltage from 5-24V. Max drain of 2.4ma when driven with 24V.

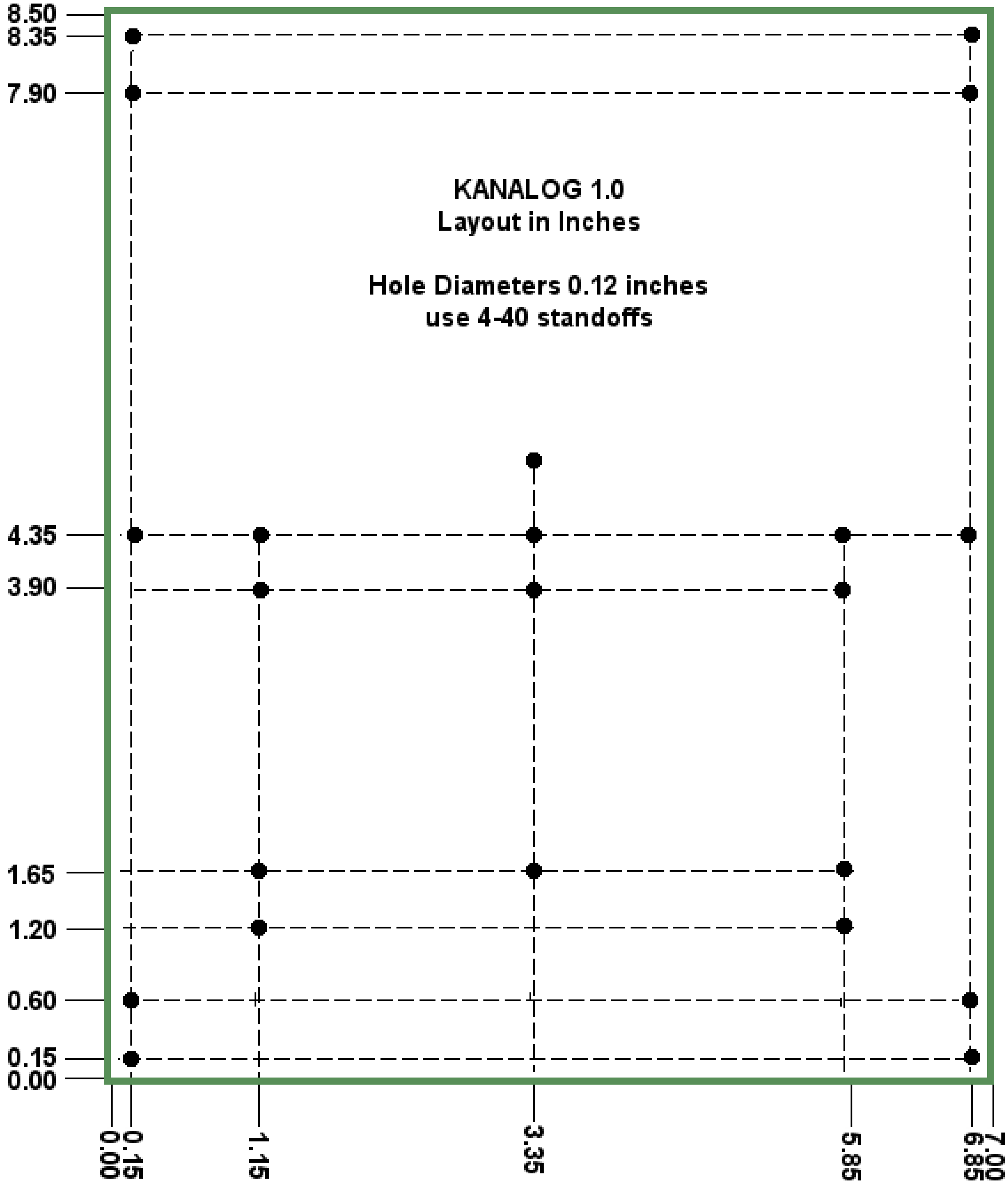


Kanalog Block Diagram



Kanalog Board Layout




KANALOG 1.0
Layout in Inches
Hole Diameters 0.12 inches
use 4-40 standoffs
8.50
8.35
7.90
4.35
3.90
1.65
1.20
0.60
0.15
0.00
0.00
0.15
1.15
3.35
5.85
6.85
7.00

Using Kanalog 1.0

Kanalog adds Analog and Digital I/O to Dynomotion's KFLOP/Kogna motion controllers.
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 (11 KHz).

Relay Drivers, Opto Inputs, Opto Outputs, Differential Encoder Inputs, LVTTL 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.

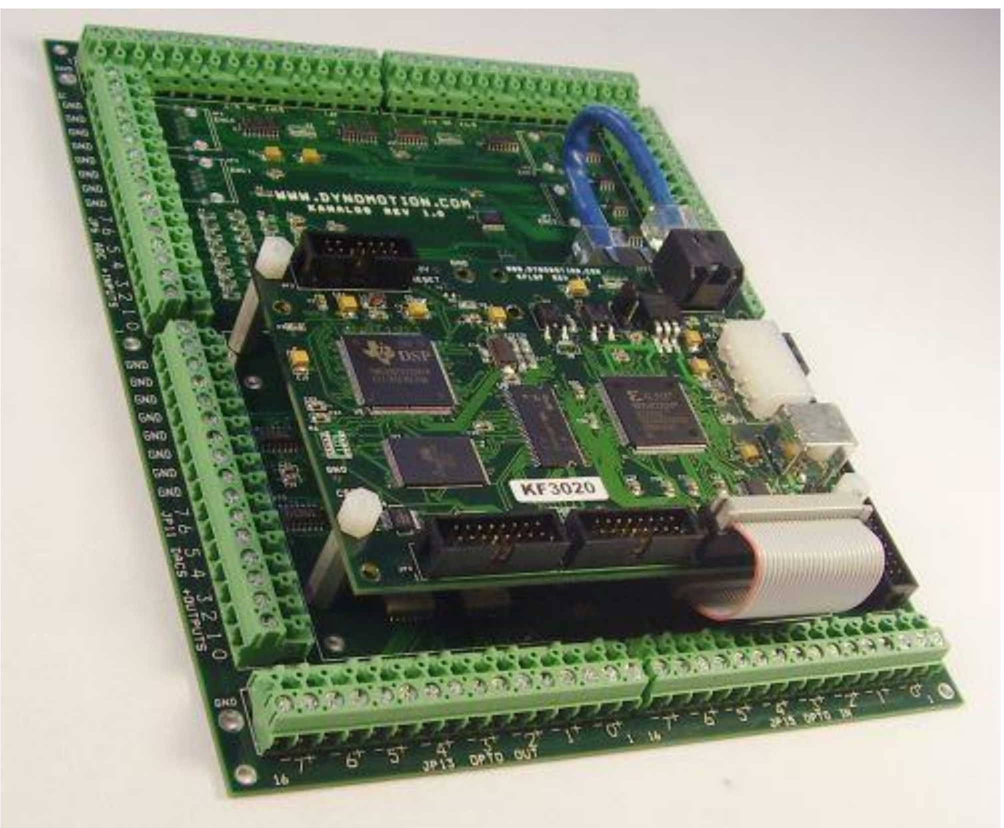
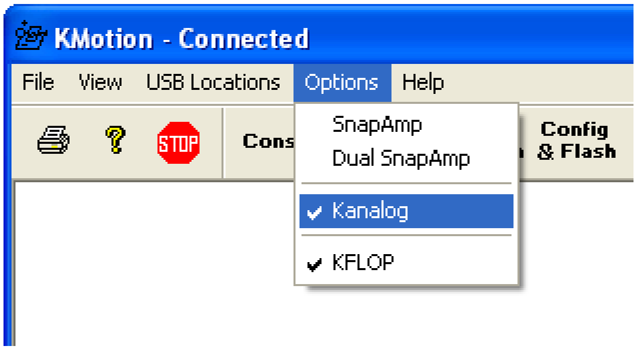


Figure 1 - Kanalog + KFLOP

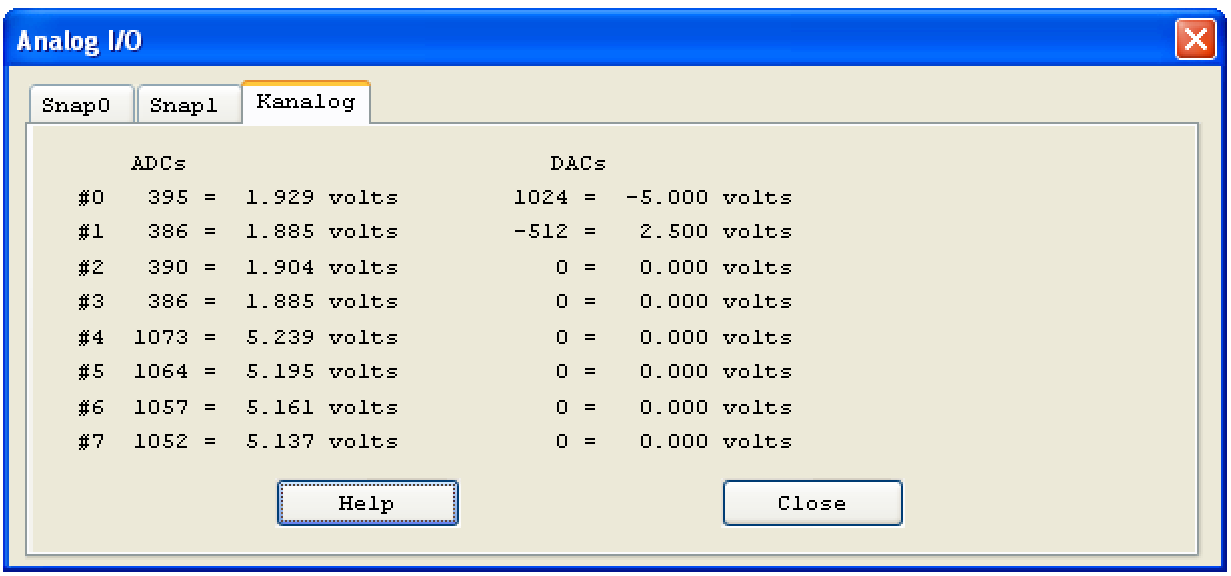
Setting Options



Set options on the KMotion Executive Program for both Kanalog and the required KFLOP.

Analog Status

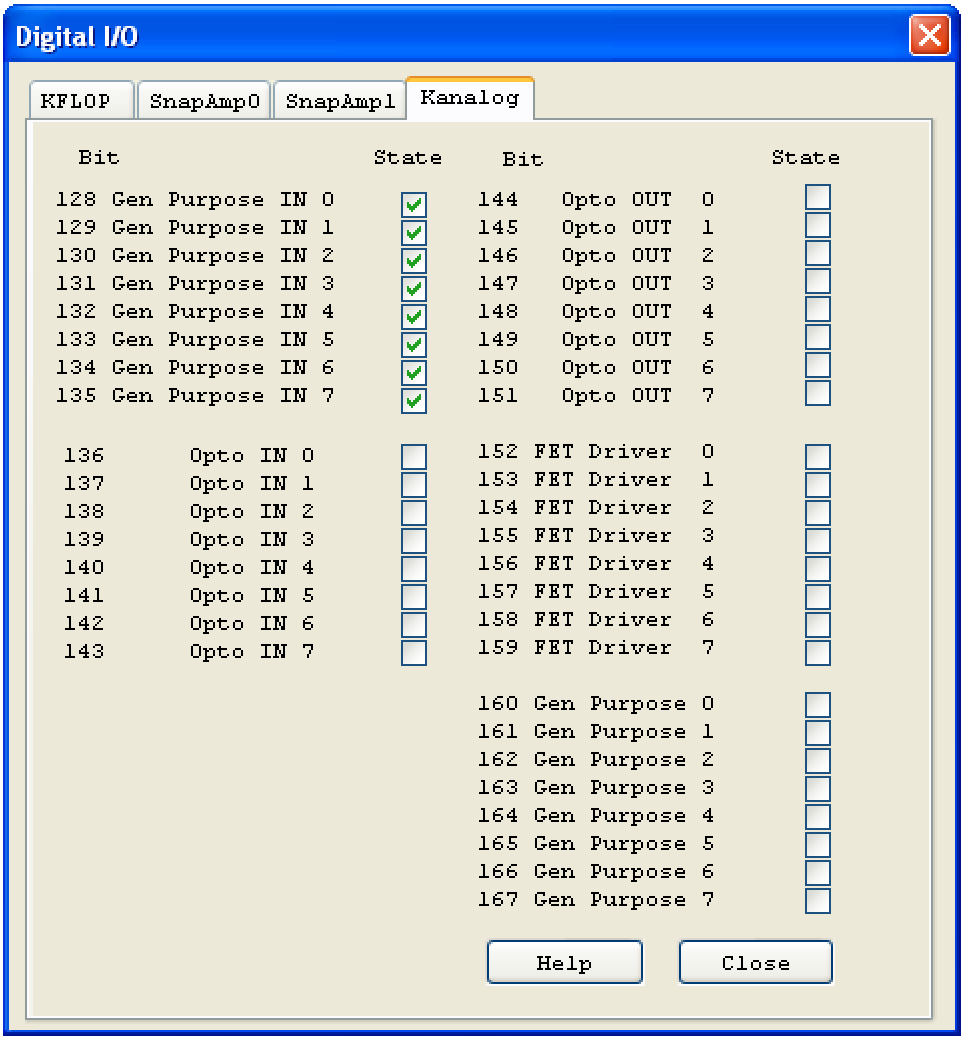
All of the Analog ADC readings and DAC settings can be observed by selecting the Kanalog Tab of the Analog I/O Screen.



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.



Example Configuration

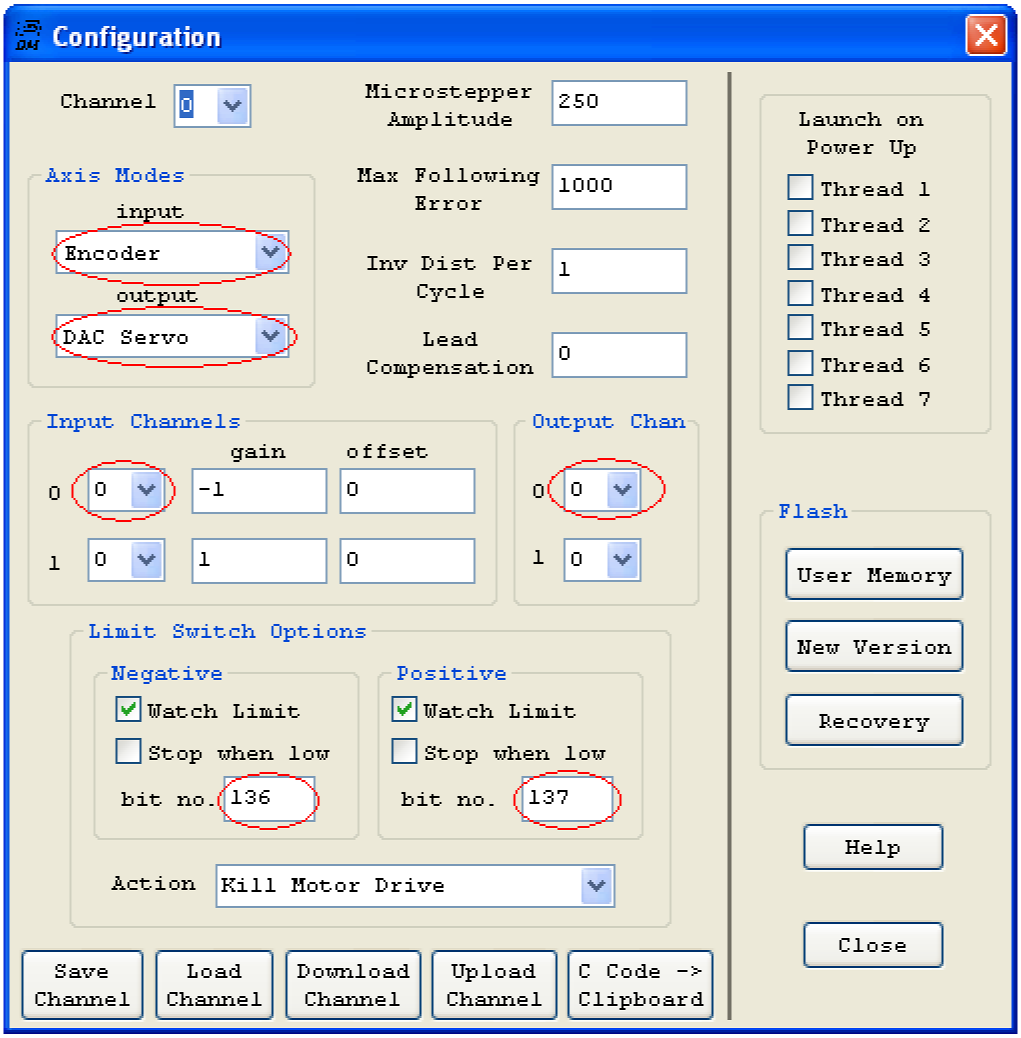
The example configuration below shows a typical configuration where an external analog motor amplifier is to be used with differential encoder feedback and optically isolated limit switches.

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).

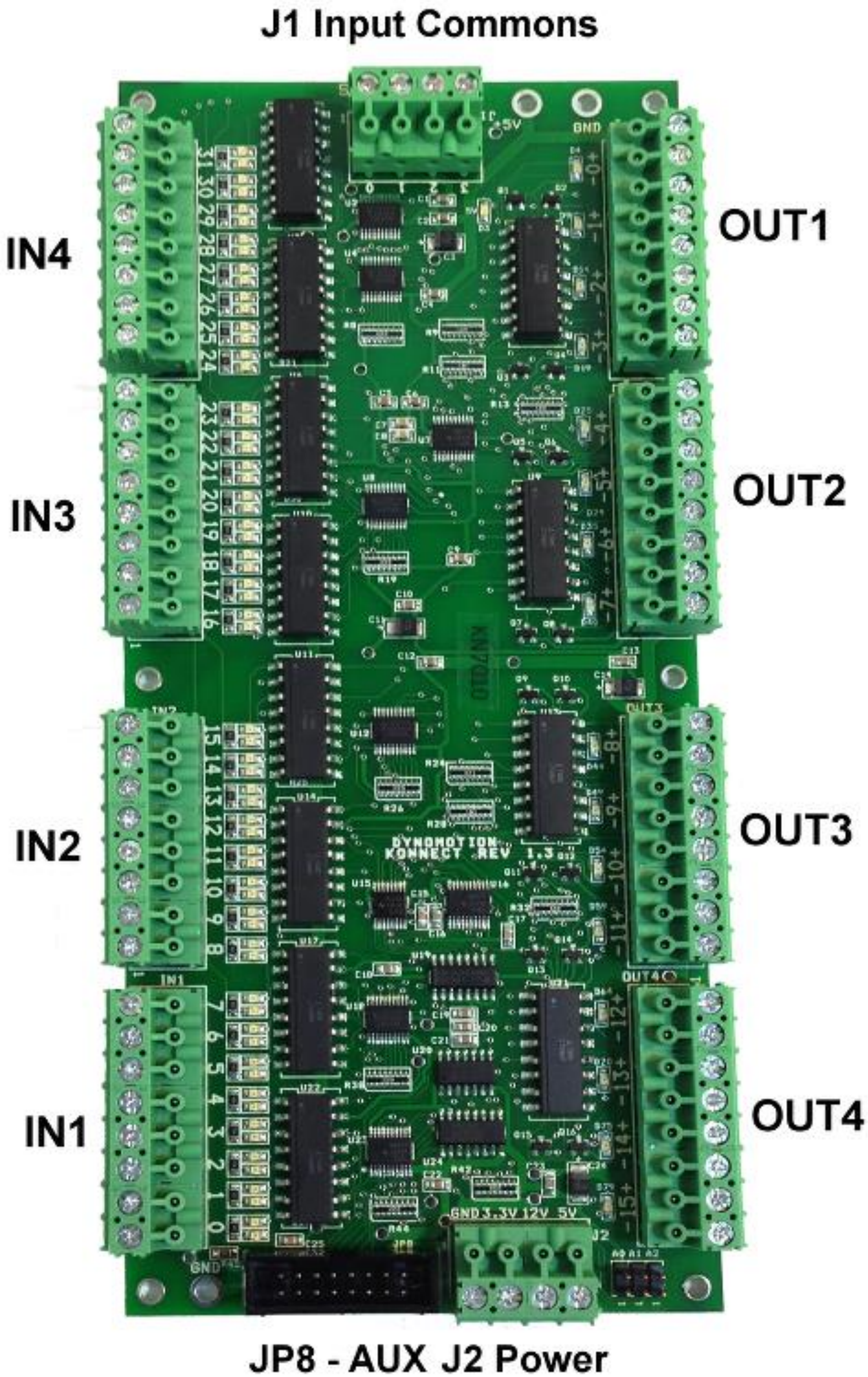
Limit Switch Options have selected I/O bits 136 and 137 which are Kanalog Opto Inputs 0 and 1 on JP15 across pins 1&2 and 3&4.


Configuration
Channel 0
Microstepper Amplitude 250
Max Following Error 1000
Inv Dist Per Cycle 1
Lead Compensation 0
Axis Modes
input
Encoder
output
DAC Servo
Input Channels
gain
offset
0 0 -1 0
1 0 1 0
Output Chan
0 0
1 0
Limit Switch Options
Negative
Watch Limit
Stop when low
bit no. 136
Positive
Watch Limit
Stop when low
bit no. 137
Action Kill Motor Drive
Save Channel
Load Channel
Download Channel
Upload Channel
C Code -> Clipboard
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

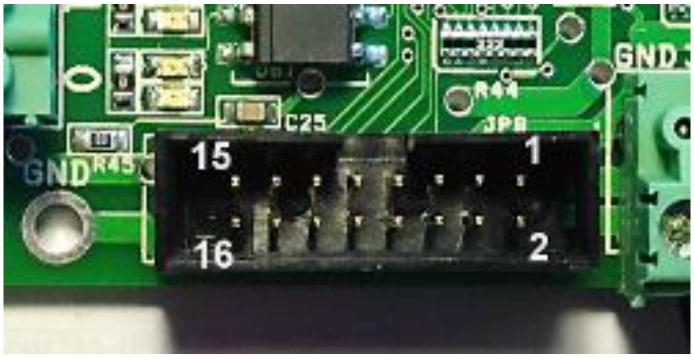
Konnect 1.3 Hardware

Function	Parameter	Specification
Inputs	Number Type V_{IH} V_{IL} Input Resistance Banks Bank Commons	32 Inputs Full Opto Isolation 4.75V min On Voltage, 25V max allowed Voltage 2V Max Off Voltage 4K Ohms 4 Banks of 8 Inputs per bank Common Anode or Cathode (Inputs activate with + or - voltage relative to Common)
Outputs	Number Type Configuration Max Voltage Max Current On Voltage Drop	16 Outputs Full Opto Isolation Darlington Transistor 30V 0.25A 0.6V max @ 2ma 0.9V max @ 0.25A
Connectors	JP8 Aux Bus to KFLOP Input Commons J2 Power IN0, IN1, IN2, IN3 OUT0, OUT1, OUT2, OUT3	16 pin Header (IDC 0.1 inch pitch) 4 pin Pluggable Screw Terminal 5mm pitch 4 pin Pluggable Screw Terminal 5mm pitch 8 pin Pluggable Screw Terminal 5mm pitch 8 pin Pluggable Screw Terminal 5mm pitch
Logic Supplies	Voltage VDD5 Max Current Voltage VDD33 Max Current Voltage VDD12 Max Current	+5V ±10% 0.4A +3.3 ±10% 0.2A (not used internally)
Environment	Operating Temperature Storage Temperature Humidity Lead	0-40º C 0-40º C 20-90% Relative Humidity, non-condensing Lead Free RoHS
Dimensions	Length Width Height	8.0 inches (203mm) 3.5 inches (89mm) 0.75 inches (19mm)

KONNECT - Connector Pinouts



Konnect JP8 Aux Bus Connection to KFLOP/Kogna JP6 or JP4



JP8 provides all internal signal and power connections to KFLOP/Kogna. See note below for how to write the code depending on which connector is used. This 16 pin ribbon connection should be as short as possible to avoid noise and crosstalk as the cable forms a high speed communication link. The Aux Bus supports multiple boards connected to the same cable. In most case the specifics of the Aux Bus will be handled internally by KFLOP/Kogna and no knowledge of the signals will be required for use.

The 8 data bits (DB0-DB7) are bi-directional. CLKIN and STARTIN allow a board to be selected by the address placed on the bus, and then a fixed sequence of 8-bit writes (2) and reads (5) can be performed using the CLKIN signal.

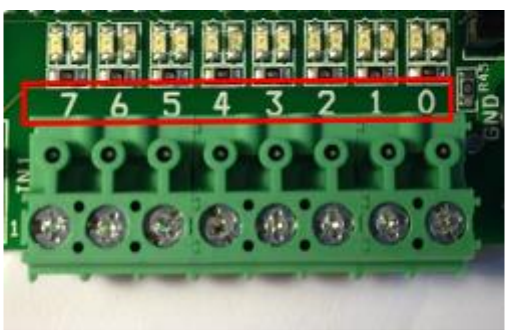
Pin	KFLOP/Kogna Name	Konnect Name
1	VDD5	VDD5
2	VDD12	VDD12
3	VDD33	VDD33
4	RESET#	RESET#
5	IO26	DB0
6	IO27	DB1
7	IO28	DB2
8	GND	GND
9	GND	GND
10	IO29	DB3
11	IO30	DB4
12	IO31	DB5
13	IO32	DB6
14	IO33	DB7
15	IO34	CLKIN
16	IO35	STARTIN

J2 Konnect Power

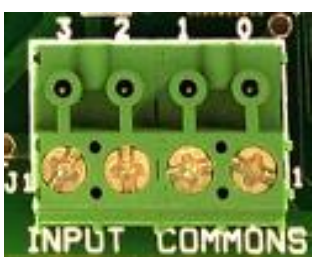
The power signals from KFLOP's/Kogna's Aux Bus are available on the J2 terminals. It is not normally necessary to apply power to these terminals as power is supplied from KFLOP/Kogna through the JP8 connector. For multiple boards it may be desirable to apply additional higher gauge power and GND connections between boards. Note the Signal labeled 12V is connected to the KFLOP/Kogna 12V signal but may not be necessarily 12V. KFLOP/Kogna and Konnect do not require or use the 12V signal but only pass the signal through the various connectors. Disk Drive Power Supplies usually supply +12V into KFLOP JR1 Pin 1. Also note that if any of these supply voltages are used for the Inputs or outputs then the Input wiring will not be isolated from KFLOP/Kogna.

It is possible to supply +5V power to KFLOP /Kogna through these terminals if the power consumption on KFLOP/Kogna is less than 1 Amp and the ribbon cable connection is short (several inches or less).

Optically isolated Inputs



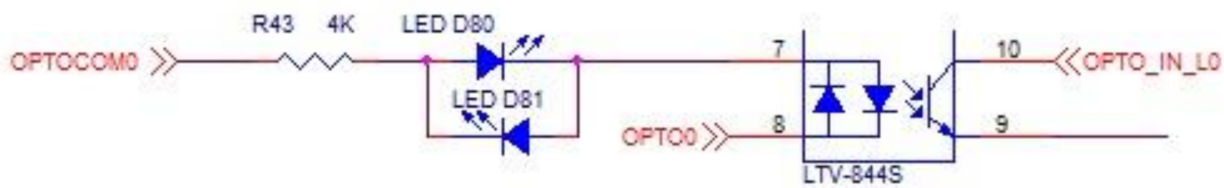
Individual Inputs (one of 4)



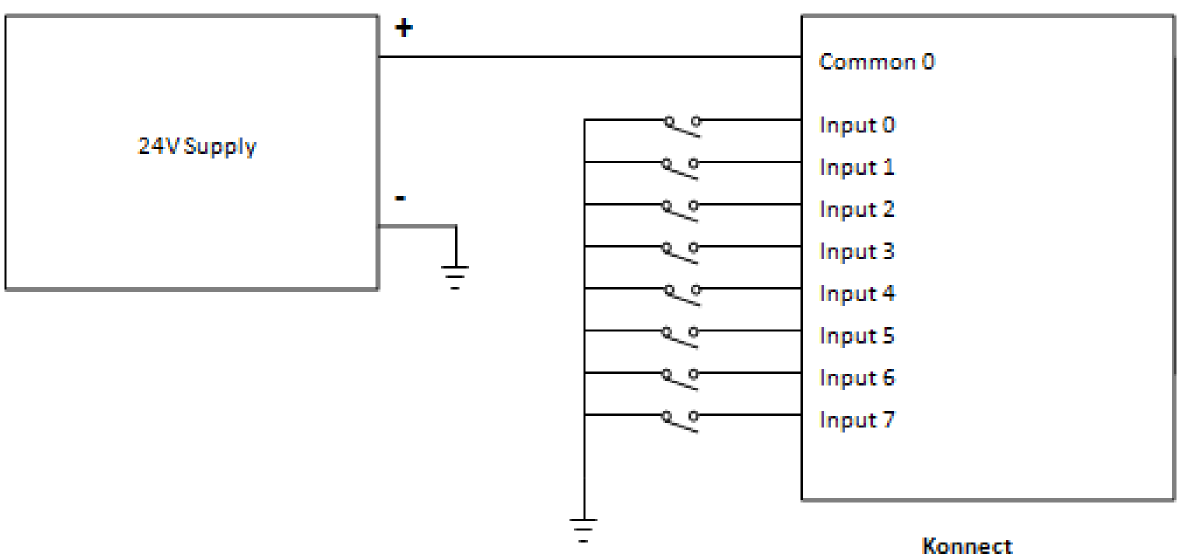
Input Bank Commons

Konnect's 32 optically isolated inputs are grouped into 4 independent Banks of 8 inputs. This allows any Bank to be used with either sourcing or sinking signals. However all inputs in the same bank share the same common and must operate in the same mode. The 4 independent Banks also allow different supply voltages and supply isolation to be used for each Bank. 24V is the preferred input voltage and will draw (24V - 1.4 - 1.4)/4K ~ 5ma of current.

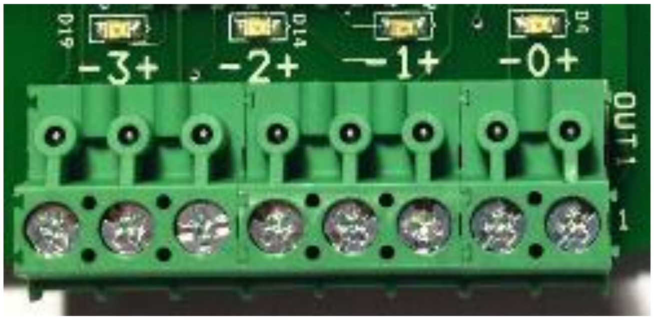
Konnect's 32 optically isolated inputs can be driven from a +/- 4.75V to a +/-25V signal. Less than 2V should be applied to ensure the input is off. One of the 32 input circuits is shown below. The input consists of a AC type of Optocoupler in series with an LED indicator in series with a 4Kohm resistor. The AC type of input allows the common to be connected to either the Positive or Negative Supply voltage so that either sinking or sourcing outputs can be used (current can flow either direction to activate the input).



External Wiring would typically be arranged such as:



Opto Isolated Outputs

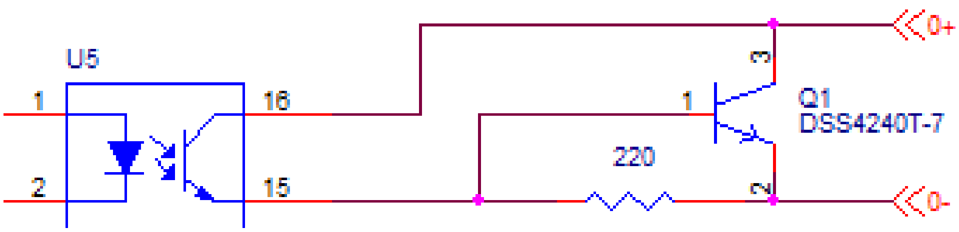


Input Bank Commons

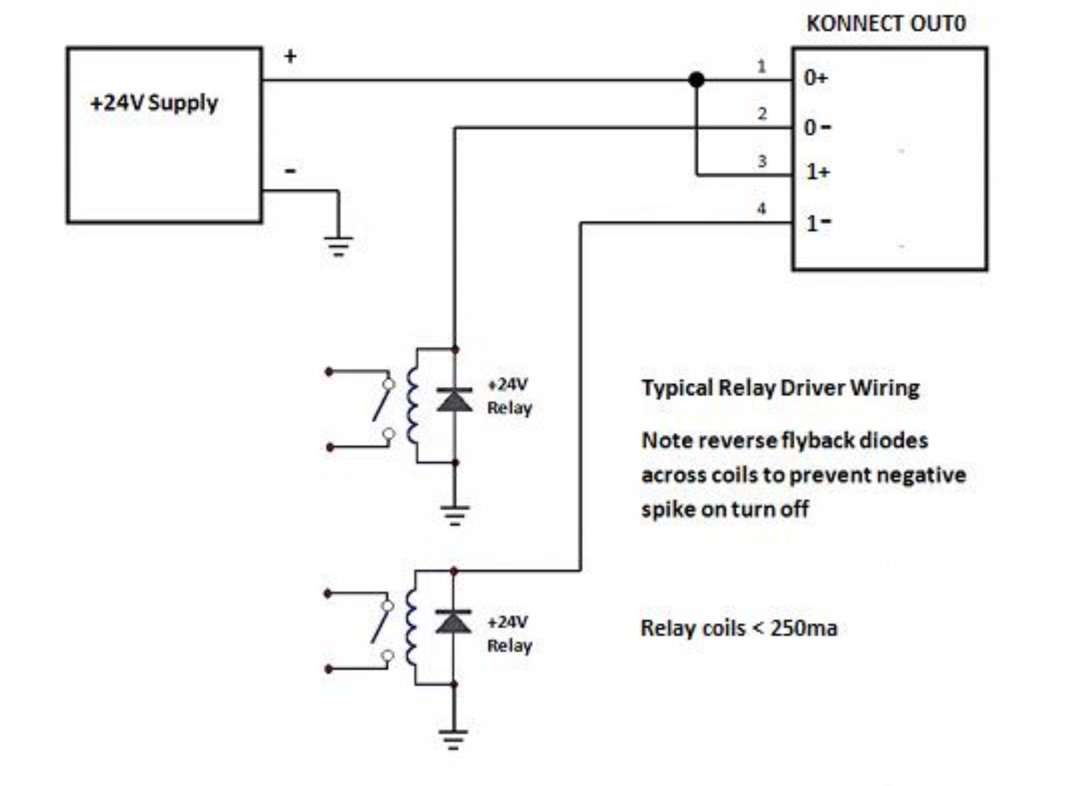
Each of the 16 Optically Isolated Outputs is completely isolated with a + and - terminal. When active the output passes current from the + to - terminals much like a relay contact would.

The Outputs are capable of driving medium power devices such as relay coils. Loads up to 30V @ 250ma may be driven.

The internal Konnect Output Circuitry is shown below which converts a photo transistor output to darlington transistor output. The 220Ohm resistor allows the output have a low on voltage for low currents before the transistor turns on. This allows the output to drive low current circuits with a smaller on voltage(2ma @ < 0.6V). For example LVTTL/TTL inputs require less than 0.8V to guarantee a low input. At higher currents Q1 becomes active and keeps the on voltage drop below 0.9V at 0.25A. Switching 0.25A at 24V allows loads up to 6W to be driven.



A typical wiring diagram driving 24V relays. Because of the less than 0.9V drop on the Konnect outputs the load will be driven with more than 23.1V.



Board Address Selection Jumpers

The KFLOP's/Kogna's Auxiliary Port works as a Bus where multiple boards can be connected to the same Port. Each board has an address so it can be selected as active. For a single Konnect board in the system removing all Jumpers will configure the Konnect Board Address as zero. If multiple boards are to be used set each board to a unique address.

A0	A1	A2	Address
Removed	Removed	Removed	0
Installed	Removed	Removed	1
Removed	Installed	Removed	2
Installed	Installed	Removed	3
Removed	Removed	Installed	4
Installed	Removed	Installed	5
Removed	Installed	Installed	6
Installed	Installed	Installed	7

KFLOP /Kogna can be configured to service each Konnect Board using the AddKonnect function. The first parameter is the board address. The second parameter is the address of where KFLOP/Kogna should obtain data to send to Konnect's 16 Outputs. The 3rd Parameter the address of where KFLOP/Kogna should place the data received from Konnect's 32 Inputs.

In most cases the addresses will be KFLOP/Kogna Virtual I/O bit locations. KFLOP/Kogna has two sets of Virtual I/O Bits, standard and extended. The standard consists of 16 Bits in VirtualBits, and the Extended consist of 1024 Bits in VirtualBitsEx[32].

The code below configures KFLOP to service 4 Konnect Boards (192 IO bits):

```
InitAux();
AddKonnect(0,&VirtualBits,VirtualBitsEx);
AddKonnect(1,VirtualBitsEx+1,VirtualBitsEx+2);
AddKonnect(2,VirtualBitsEx+3,VirtualBitsEx+4);
AddKonnect(3,VirtualBitsEx+5,VirtualBitsEx+6);
```

Board 0 has Output Bits mapped to 48-63 and Input Bits Mapped to 1024-1055
Board 1 has Output Bits mapped to 1056-1071 and Input Bits Mapped to 1088-1119
Board 2 has Output Bits mapped to 1120-1135 and Input Bits Mapped to 1152-1183
Board 2 has Output Bits mapped to 1184-1199 and Input Bits Mapped to 1216-1247

Typical IO Mapping for Standard Single Konnect

Note: Konnect(s) can be connected to JP6 (Aux1):

```
InitAux();
AddKonnect(0,&VirtualBits,VirtualBitsEx);
```

Or JP4 (Aux0) but use:

```
InitAux();
AddKonnect_Aux0(0,&VirtualBits,VirtualBitsEx);
```

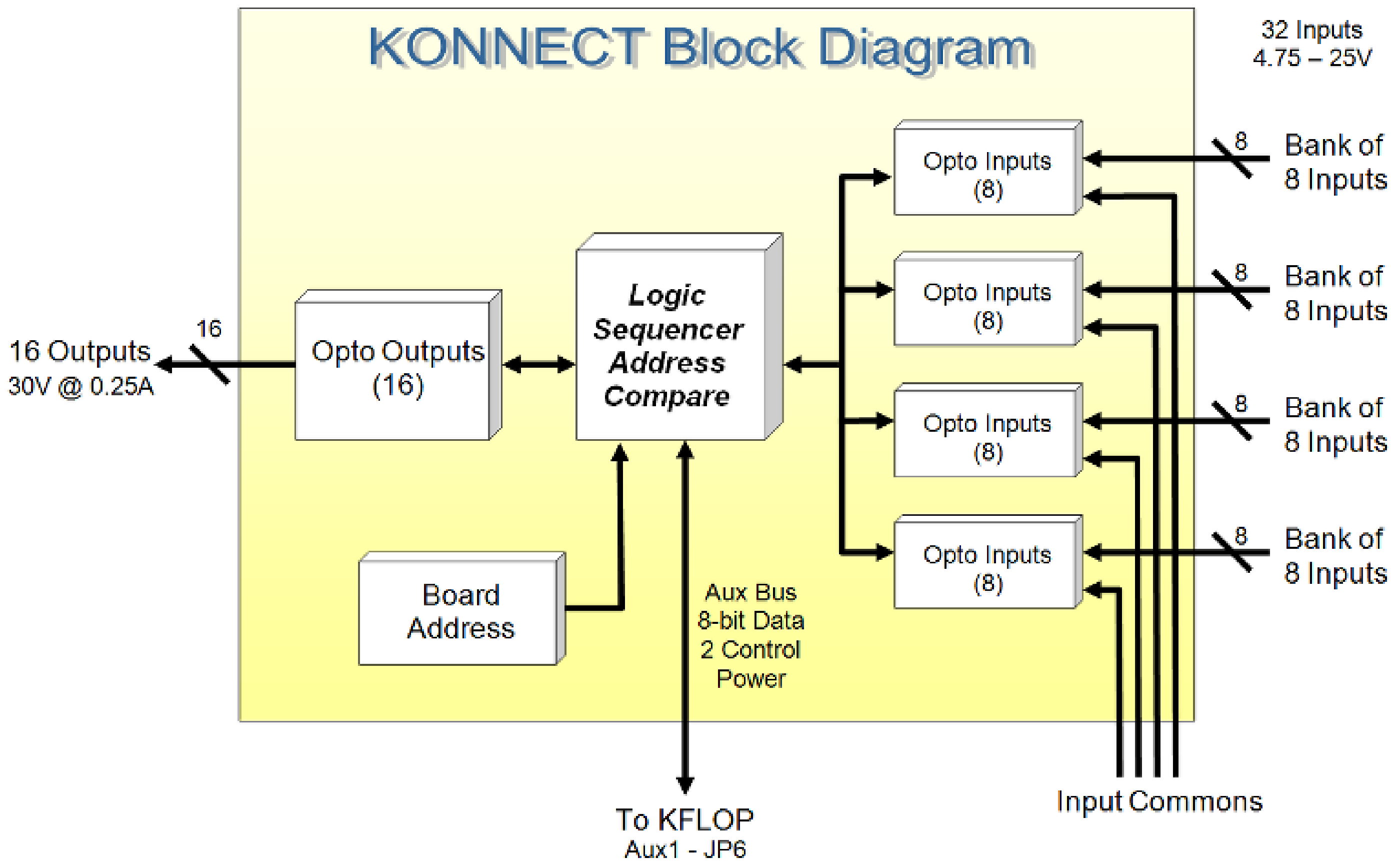
Outputs	Virtual IO Number
0	48
1	49
2	50
3	51
4	52
5	53
6	54
7	55

8	56
9	57
10	58
11	59
12	60
13	61
14	62
15	63

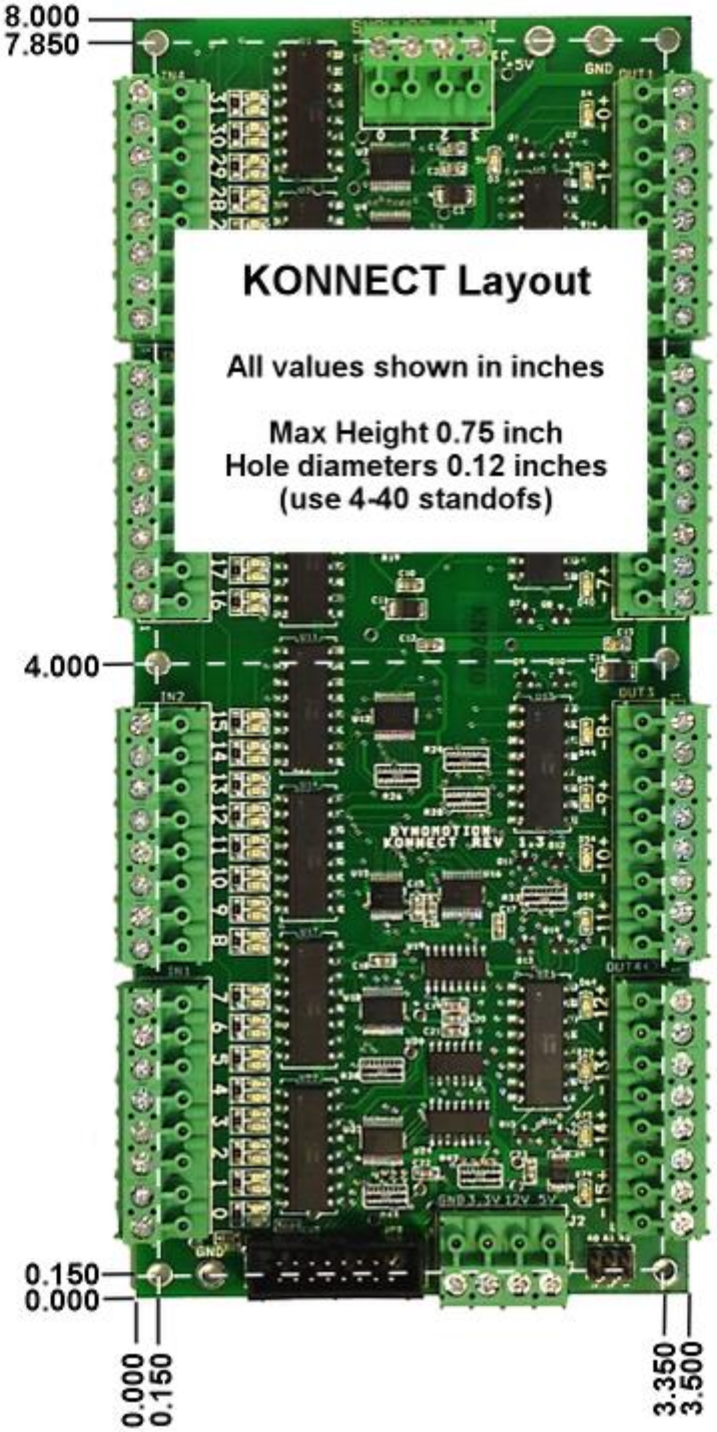
Inputs	Virtual IO Number
0	1024
1	1025
2	1026
3	1027
4	1028
5	1029
6	1030
7	1031
8	1032
9	1033
10	1034
11	1035
12	1036
13	1037
14	1038
15	1039
16	1040
17	1041
18	1042
19	1043
20	1044
21	1045
22	1046
23	1047
24	1048
25	1049
26	1050

27	1051
28	1052
29	1053
30	1054
31	1055

KONNECT Block Diagram



KONNECT Board Layout



KONNECT - PWM to Analog Example

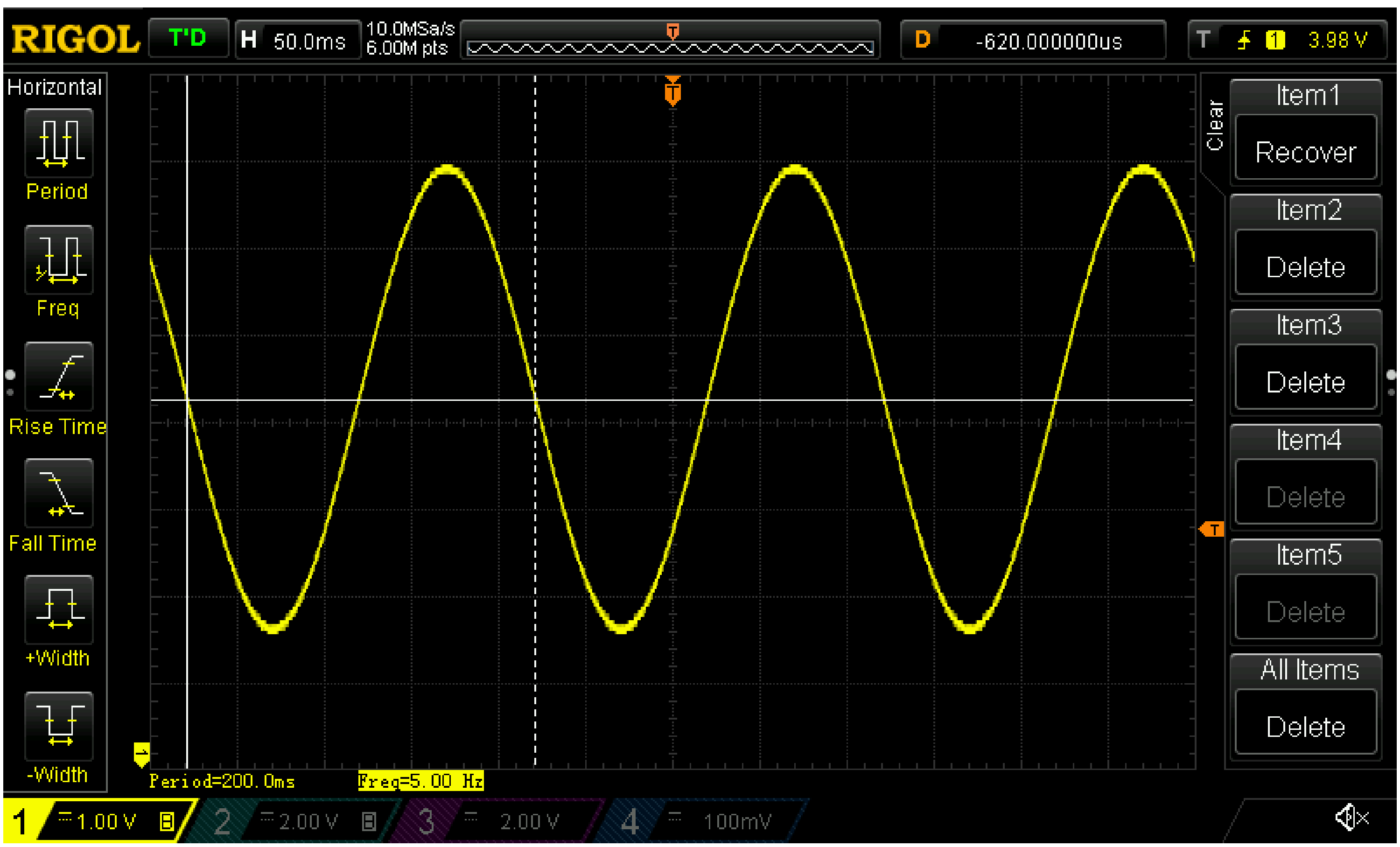
KFLOP/Kogna+Konnect+Simple Filter Circuit can be used to produce a programmable analog signal. This might be used as a basic Spindle Speed Control Signal if no others are available.

This demonstrates the flexibility of the Konnect Outputs which are Isolated, can sink or source current, fast, medium power, and low impedance

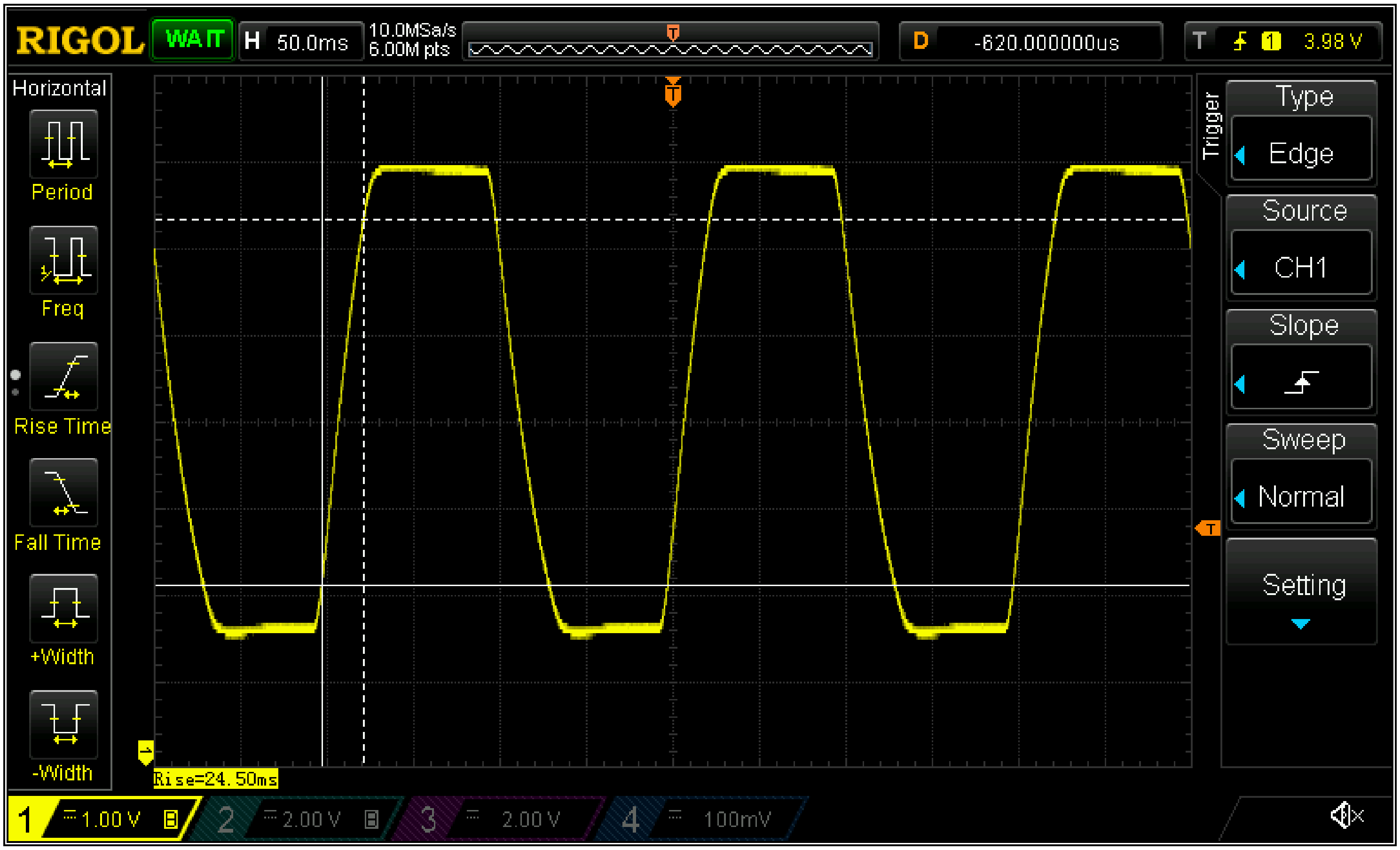
Results

Analog Oscilloscope traces of Analog signals generated by a KFLOP/Kogna Software generated PWM Signal controlling two optically isolated Konnect Outputs that are then passed through a simple low pass filter.

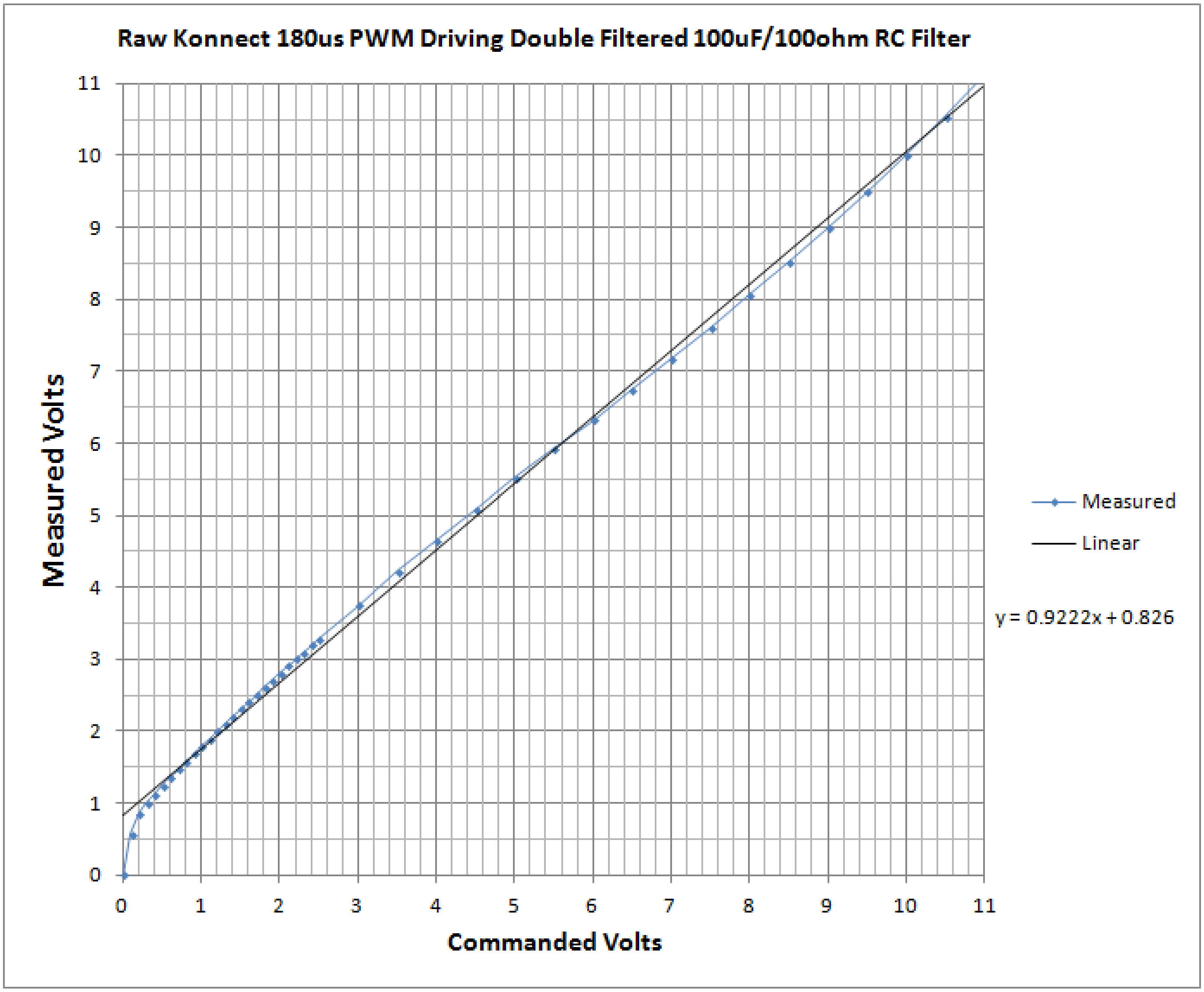
High resolution and response rates easily usable for speed control applications



Square wave to test large signal changes. Significant change in < 25ms



Reasonable linearity. This would require calibration if higher accuracy is needed.



Circuit

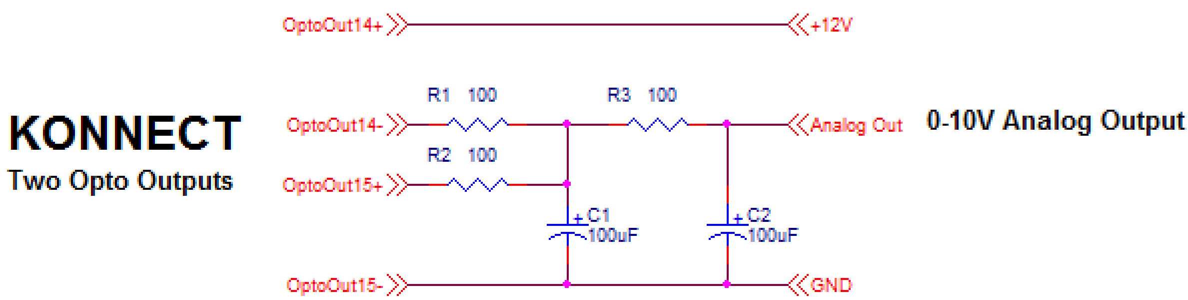
Simple cascaded dual low pass filters. Only 5 components. One resistor type and one capacitor type. Component values are not critical (R1 and R2 should be matched).

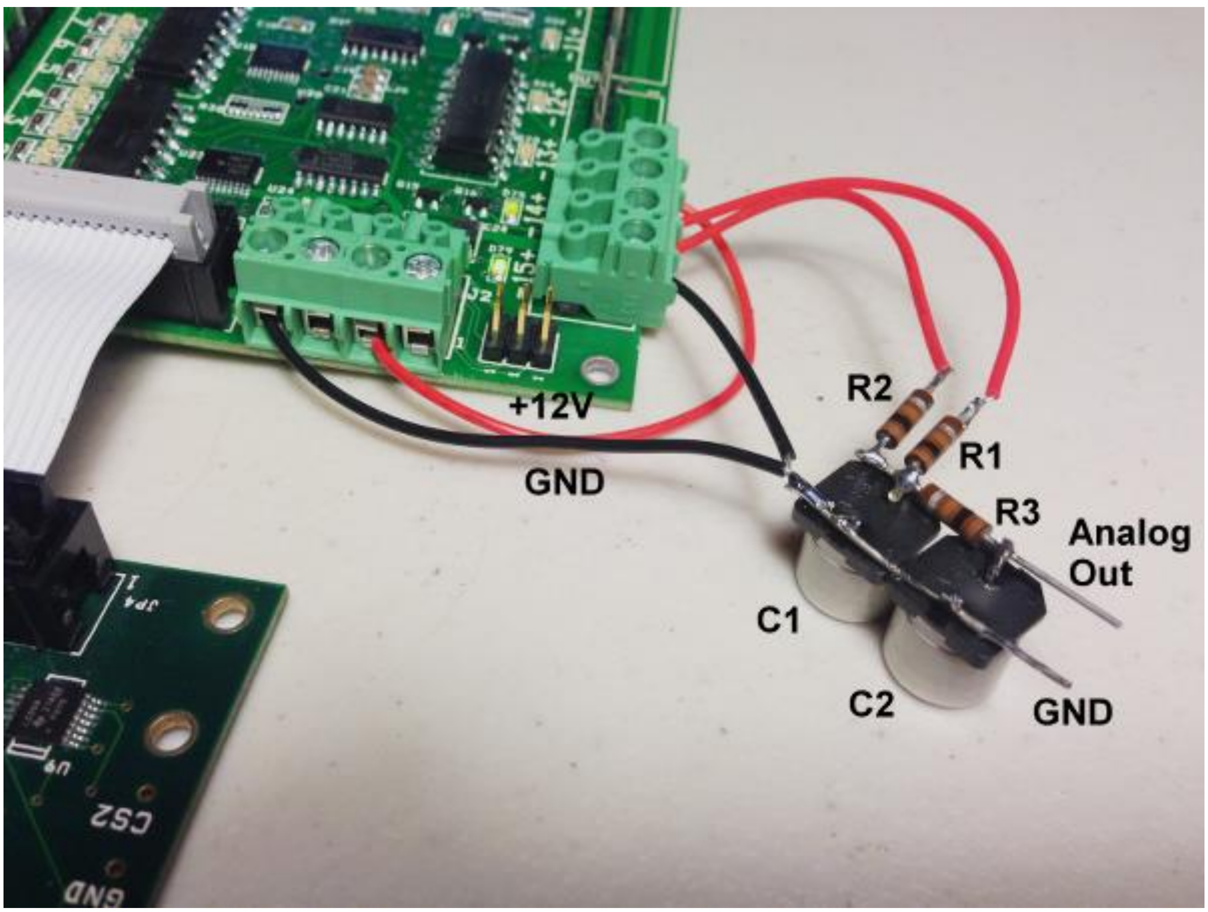
One Konnect Output charges the capacitors and one output discharges them. Both should not be turned on simultaneously (but no damage will occur if they are as R1+R2 will limit the current).

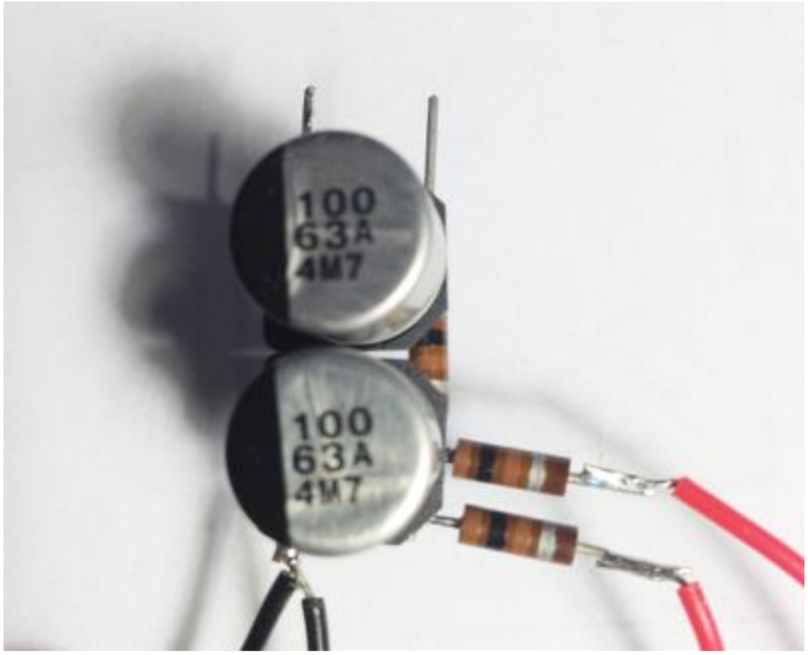
The relatively low resistance values (100 Ohms) provides low output impedance so that any connected load should have a minimal effect. A load of 10Kohms or higher should work well.

Double filtering provides low output ripple, while still having relatively quick response to changes, uses reasonably small capacitors, even with relatively low PWM rates.

Konnect Outputs can be updated every 180us. So 180us is the basic PWM quantum. This results in ~ 10mV p-p ripple..







Software

This software example simulates how an RC circuit would respond to an applied, switched, high/low voltage.

If the simulated voltage is below the desired output voltage then the output is switched high to charge up the capacitor, otherwise it is switched low to discharge the capacitor.

The same state that is simulated is also sent to the Konnect Outputs to drive the real circuit. The real circuit is a bit more complex but the simple model works reasonably well. The two RC circuits will eventually evolve to the same voltage as the average PWM Voltage in the steady state. Only the transient response will be slightly different. Similarly, somewhat incorrect component values will only affect the transient response. The progra values were adjusted to get the best response.

The #define statements may require changes for your specific circuit and I/O Bit used.

The Vout value is coded to create a sine wave, but more typically the value would be passed in through a global persist variable as a Spindle Speed Setting.

```
#include "KMotionDef.h"


// Enables a Konnect on KFLOP JP4 Aux Port then
// PWM's two outputs as push-pull drivers such that
// when low passed filtered with an RC circuit becomes
// a variable analog source.
//
```

```
// Configure KFLOP to service Konnect 32 Input 16 output IO board
// Board address is 0,
// 16 Outputs are mapped to Virtual IO 48-63 (VirtualBits)
// 32 Inputs are mapped to Virtual IO 1024-1055 (VirtualBits[0])
//
// Attach Service to Aux0 Port (KFLOP JP4) instead of standard Aux1 Port
(KFLOP JP6)
//

void ServiceKonnectPWM(void);

double T,T0=0;
float Vout=0.0;  // desired voltage

main()
{
    InitAux();
    AddKonnect_Aux0(0,&VirtualBits,VirtualBitsEx);

    for(;;)
    {
        T=WaitNextTimeSlice();
        ServiceKonnectPWM();

        // Fixed
        //              Vout = 0.1;


        //Generate a 5 Hz 3V Sine Wave
        Vout = 3.0f*sin(T * TWO_PI * 5.0) + 5.0;


        //Generate a Saw Tooth wave
        //              Vout = 2 + 6.0* (5.0*T - ((int)(5.0*T)));

        //Generate a 5 Hz Square wave
        //              Vout = (5.0*T - ((int)(5.0*T))) > 0.5 ? 8 : 2;
    }
}

#define C 0.00029f // 1000uF
#define R 100.0f // 100 ohms
#define Vcc 11.230f // supply voltage
#define HIGH_BIT 62 // This output drives Cap high
#define LOW_BIT  63 // This output drives Cap low


void ServiceKonnectPWM(void)
{
```

```
    static int FirstTime=TRUE;
    static float Vc=0.0f;
    static double T0;
    static int State;
    double T=Time_sec();

    if (FirstTime)
    {
        FirstTime=FALSE;
        T0=T;
        State=0;
    }
    else
    {
        float V,I;

        // Compute Voltage applied to Cap
        V=Vcc*State;

        // Compute current
        I=(V-Vc)/R;

        // Compute new Cap Voltage
        Vc += I/C*(T-T0);

        // determine next state

        if (Vc > Vout)
        {
            ClearBit(HIGH_BIT);
            SetBit(LOW_BIT);
            State=0;
        }
        else
        {
            ClearBit(LOW_BIT);
            SetBit(HIGH_BIT);
            State=1;
        }



     T0=T; // save time when applied
    }
}
```

Using SnapAmp 1000

SnapAmp is a very high performance, feature rich, efficient amplifier that expands the capability of our KFLOP/Kogna Motion Control Boards.

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

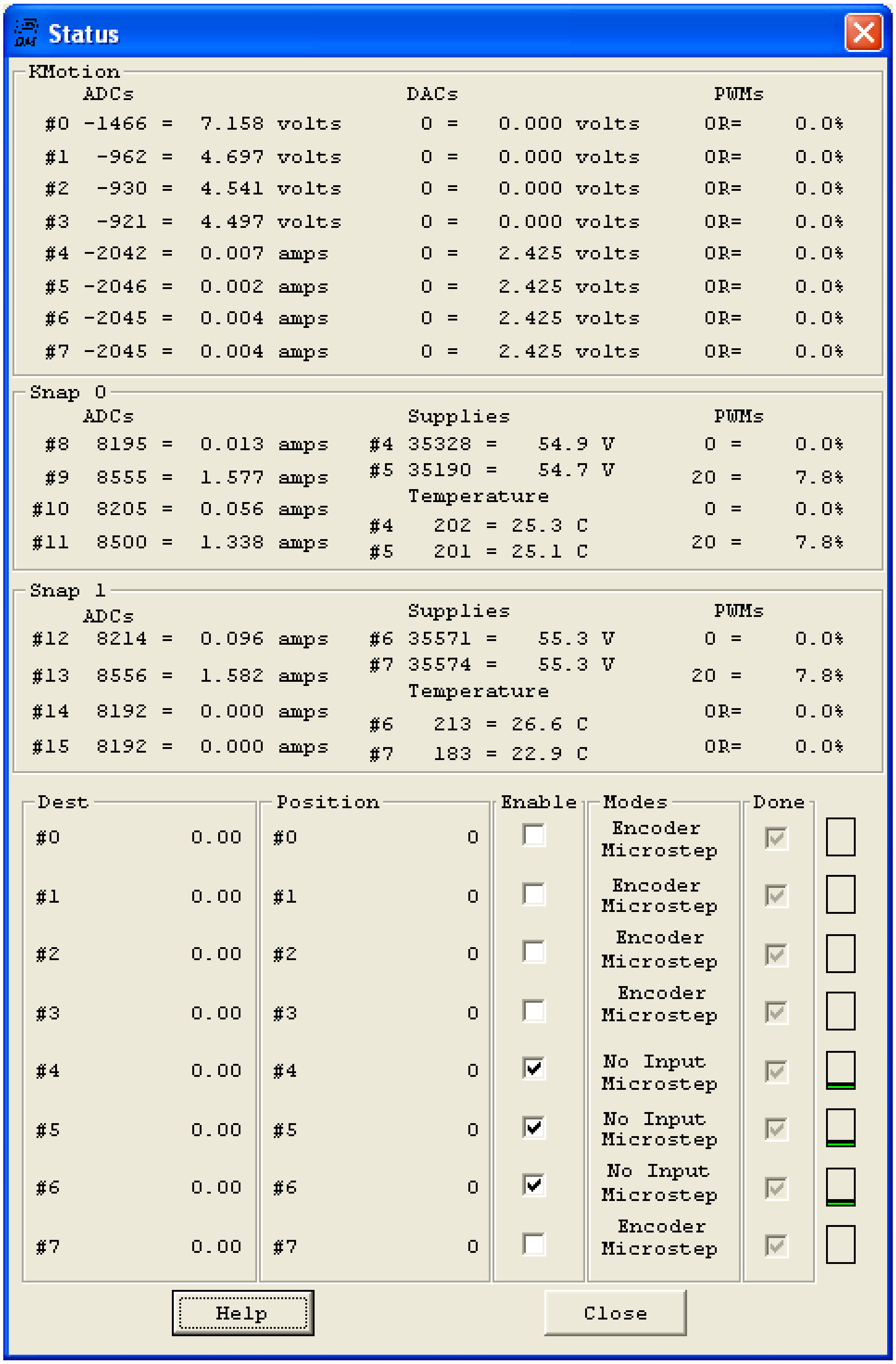
Up to two SnapAmps may be added to a Single KFLOP/Kogna Controller.

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.

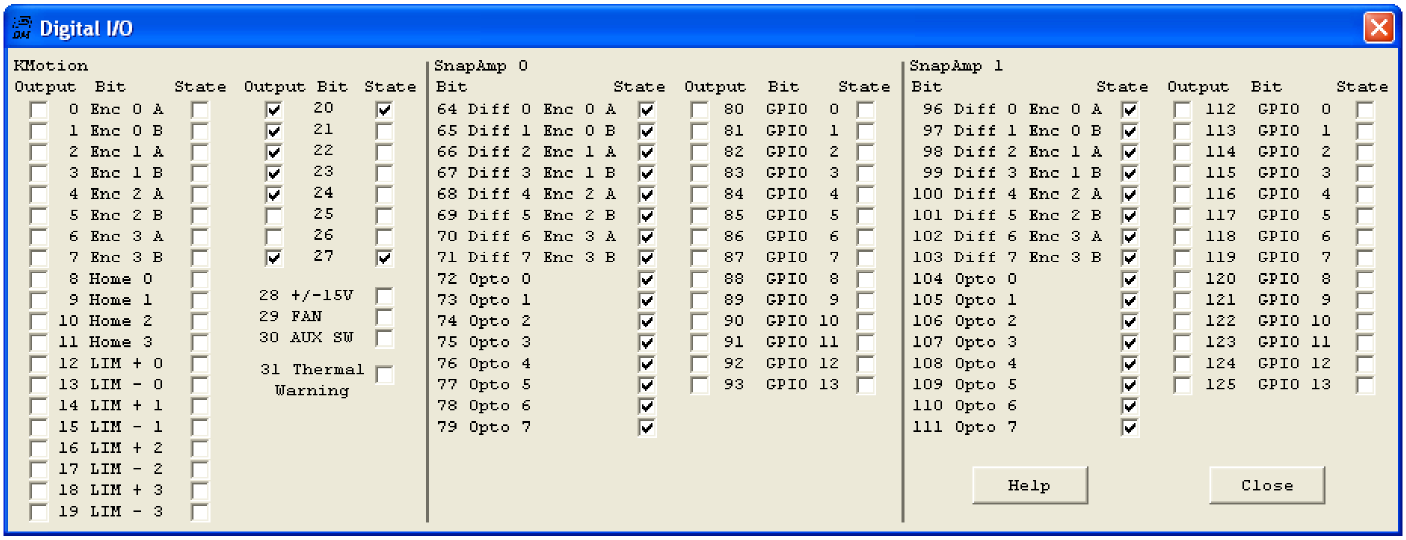
Within the KMotion executive program under the option menu set whether there are one or two SnapAmps connected to KFLOP/Kogna. This enables expanded Analog and Digital Screens that will then display the additional I/O available on the SnapAmp(s).

The middle portion of the Analog Status Screen displays the measured currents, supply voltages, temperatures, and current PWM settings.



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 KFLOP/Kogna I/O bits remain on the left portion of the screen. Note that KFLOP/Kogna I/O

Bits 20-28 are used for the high speed communication to the SnapAmps and may not be used as user IO.



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 KFLOP/Kogna 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);
}
```

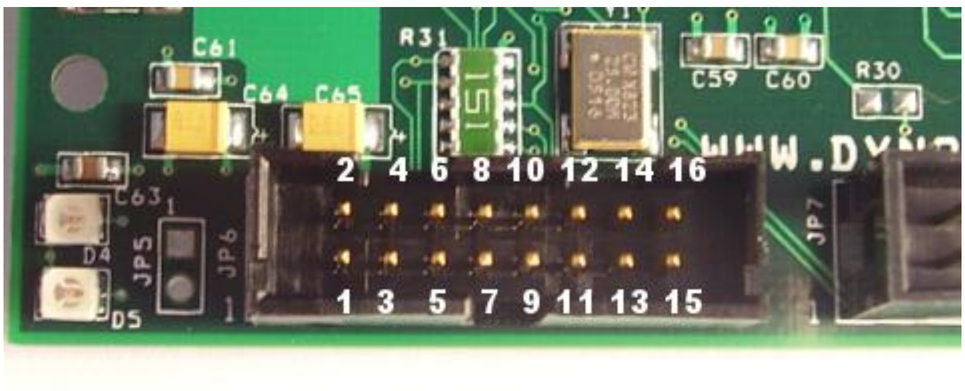
SnapAmp - Connector Pinouts

SnapAmp contains three connectors labeled JP1, JP6, and JP7.

JP6 – KFLOP/Kogna Communication & Logic Power

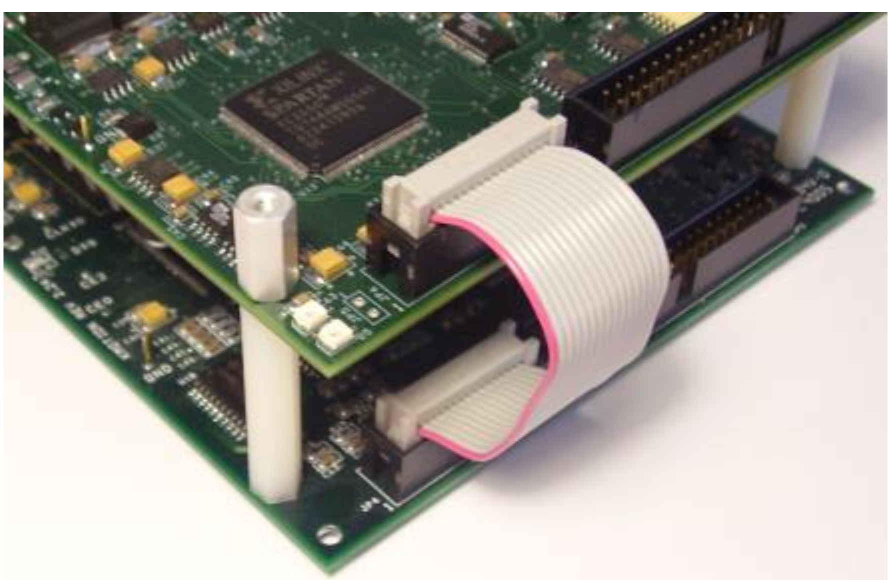
JP6 is a proprietary high-speed communication bus where command and status communication between KFLOP/Kogna and the SnapAmp Amplifier. Up to two SnapAmps may be attached to a single KFLOP/Kogna. The second of the two SnapAmps must have a configuration jumper installed into JP5.

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 KFLOP/Kogna.

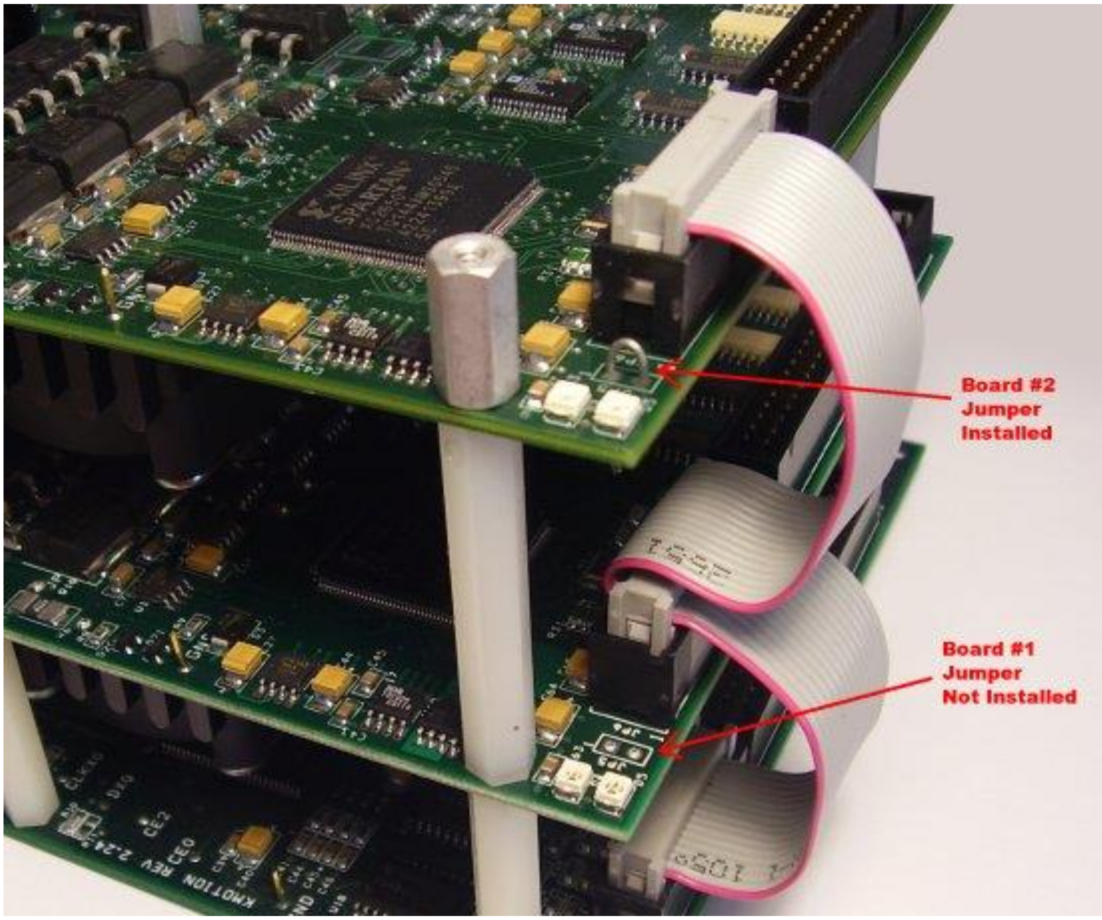
Note: 4 - 40 x 1 3/8 inch standoffs are used between SnapAmp (top) and KFLOP/Kogna (bottom).



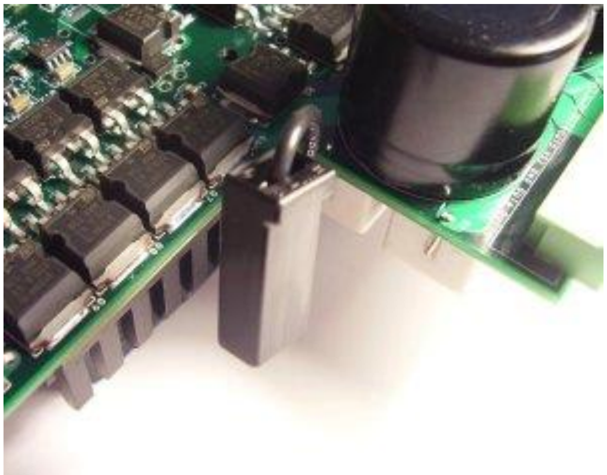
16 pin ribbon cable connection between Dual SnapAmps and KFLOP/Kogna.

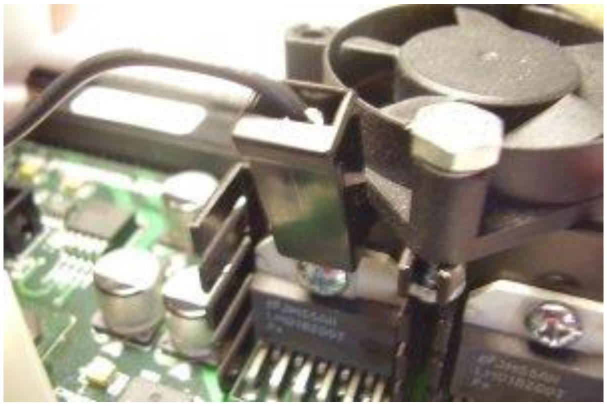
Notes:
4 - 40 x 1 3/8 inch standoffs are used between SnapAmp and KFLOP/Kogna
4 - 40 x 1 5/8 inch standoffs are used between SnapAmps

JP5 Jumper installed configures a SnapAmp as the 2nd SnapAmp.

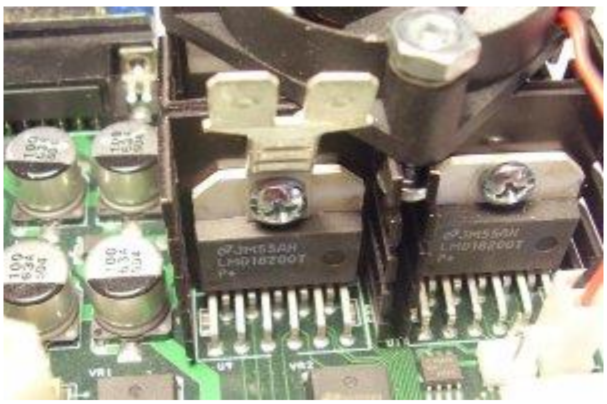


When attaching SnapAmp to KFLOP/Kogna first attach main ground plug as shown.



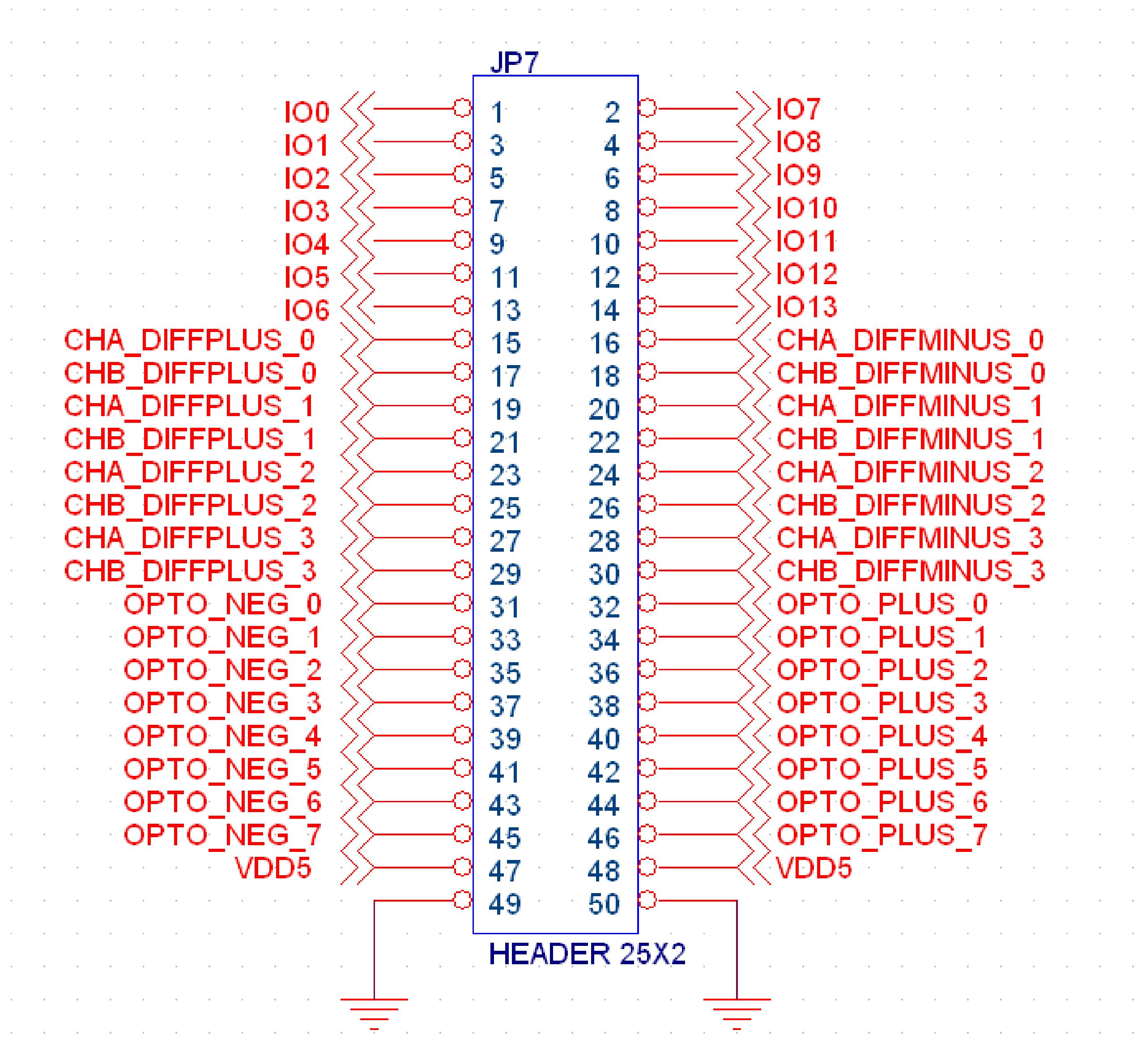


For Dual Snap Amp systems attach the 1:2 spade Y adapter before connecting the first SnapAmp.

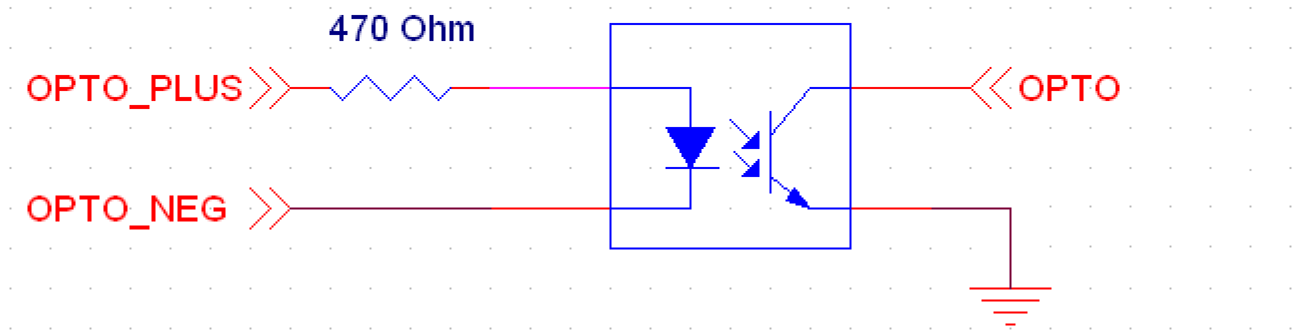


JP7 - I/O - General Purpose LVTTL - OPTO Isolated - Differential - Encoder Inputs

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. SnapAmp Opto inputs have negative true polarity (energizing the Opto is read as a Logic 0 in KFLOP).

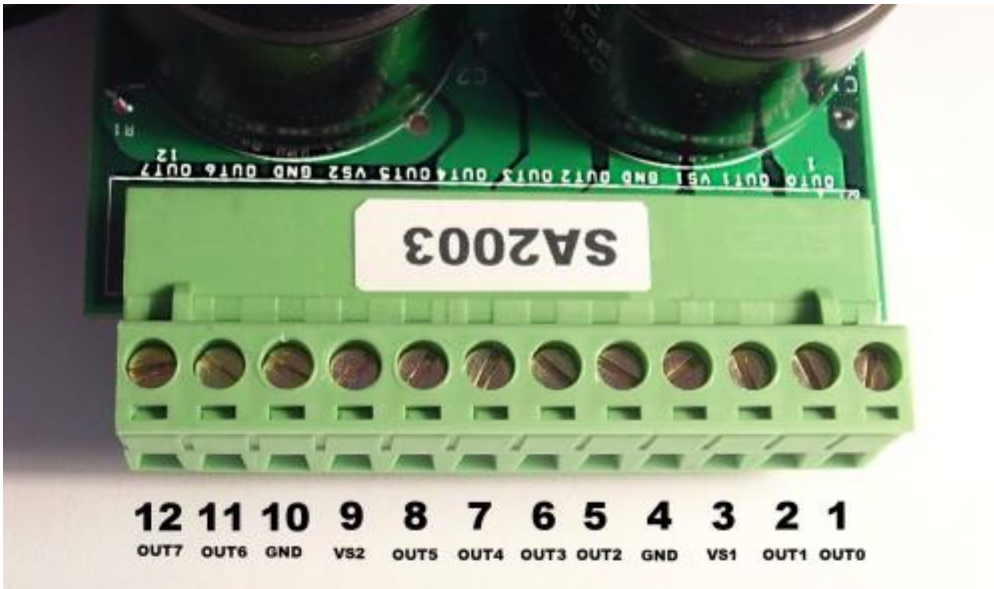


Pin	Name	Description	Bit (SnapAmp #0)	Bit (SnapAmp #1)
1	GPIO0	Gen Purpose LVTTL	80	112
2	GPIO1	Gen Purpose LVTTL	81	113
3	GPIO2	Gen Purpose LVTTL	82	114
4	GPIO3	Gen Purpose LVTTL	83	115
5	GPIO4	Gen Purpose LVTTL	84	116
6	GPIO5	Gen Purpose LVTTL	85	117
7	GPIO6	Gen Purpose LVTTL	86	118
8	GPIO7	Gen Purpose 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
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

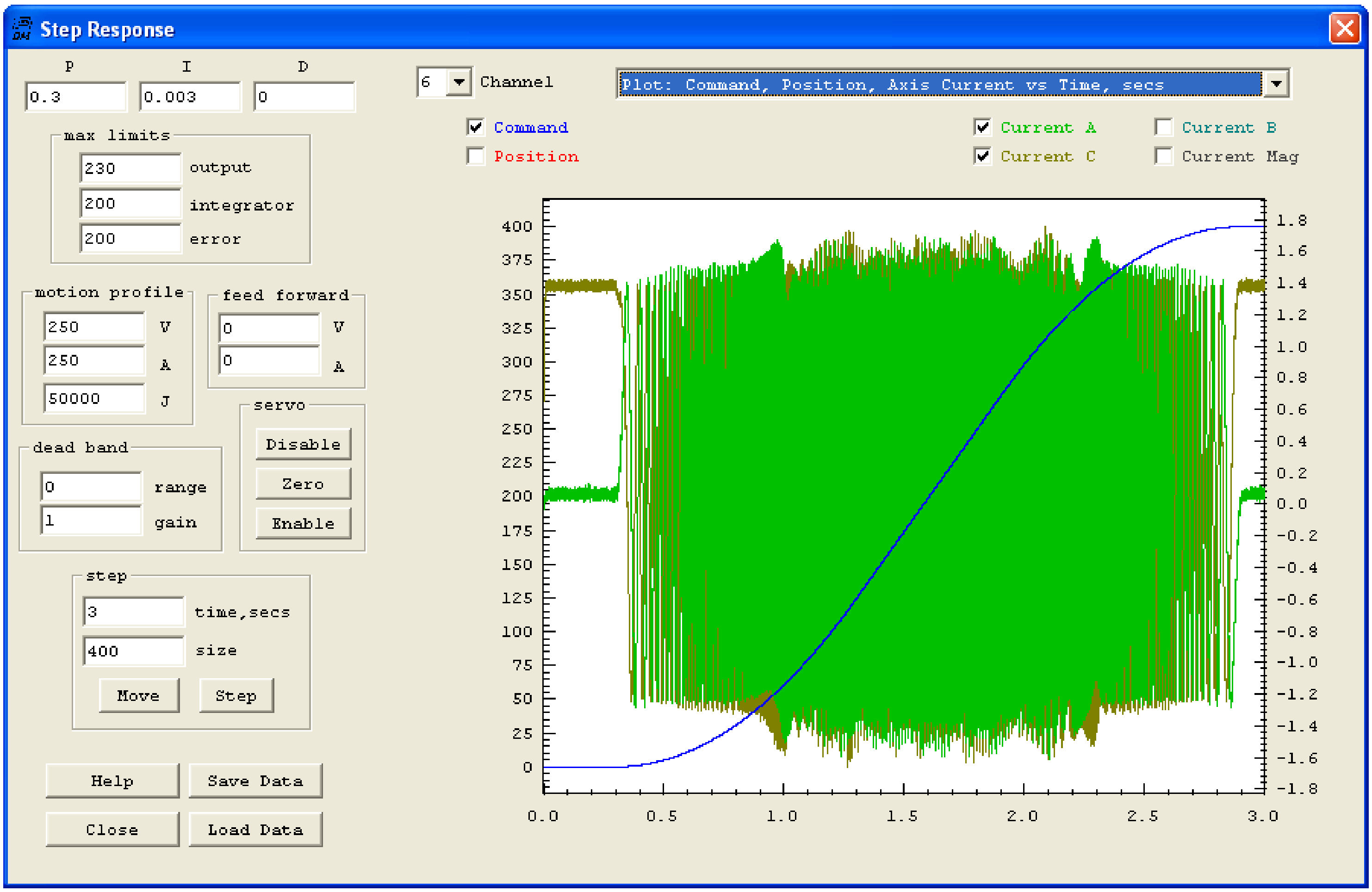


	Motor type DC - Brush	Motor type - 3 Phase brushless	Motor type - Stepper
Axis 0	Connect Motor across OUT0-OUT1 Specify PWM Channel 8	Connect Phase A to OUT0 Connect Phase B to OUT1 Connect Phase C to OUT2 Leave OUT3 disconnected Specify PWM Channel 8	Connect Coil A across OUT0-OUT1 Connect Coil B across 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		

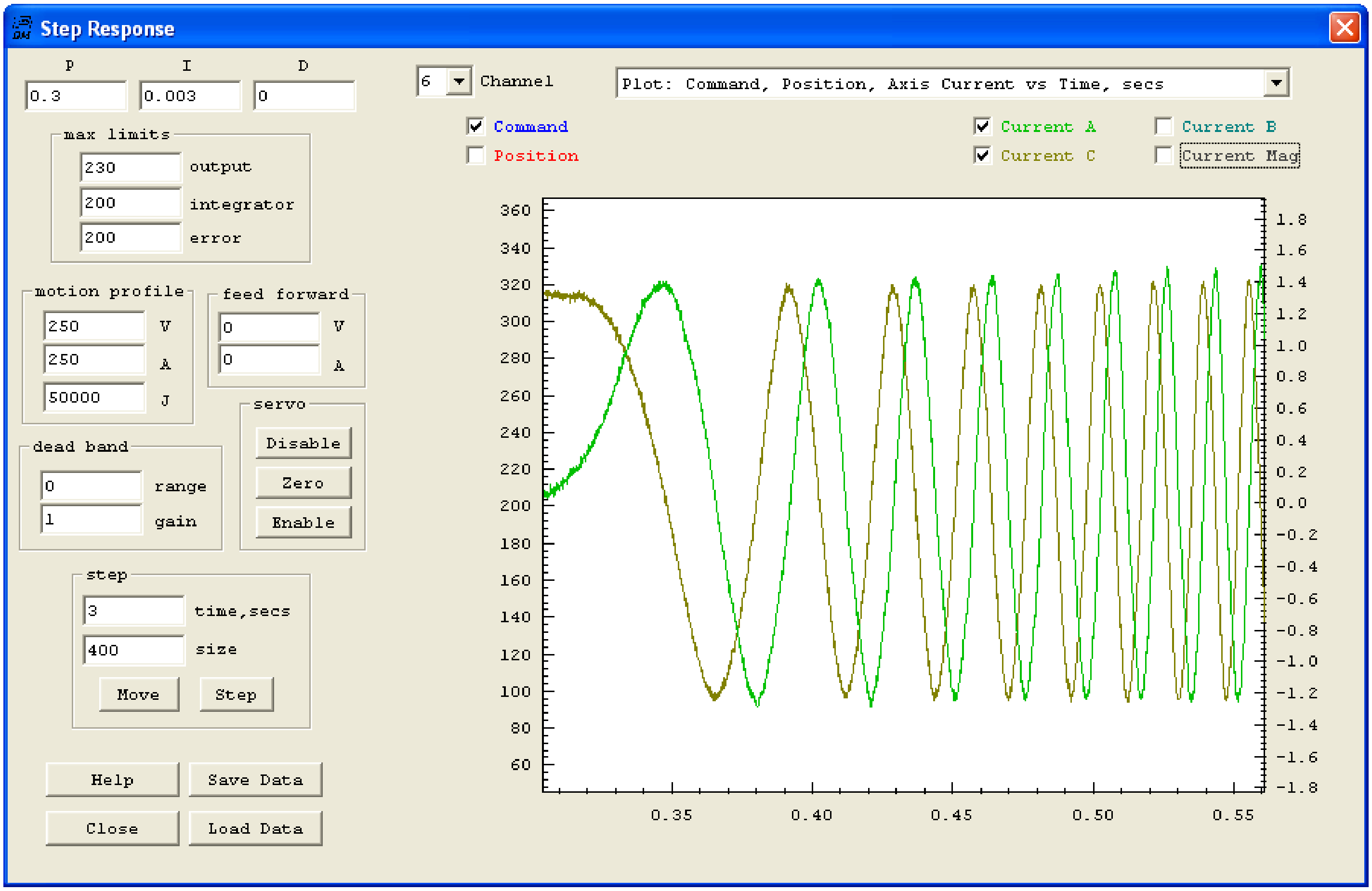
SnapAmp Stepper Motor Diagnostic Tutorial

Shown below is a plotted motion of 400 cycles (1600 full steps) with quite gradual acceleration up to a velocity of 255 cycles/sec (1000 sps)

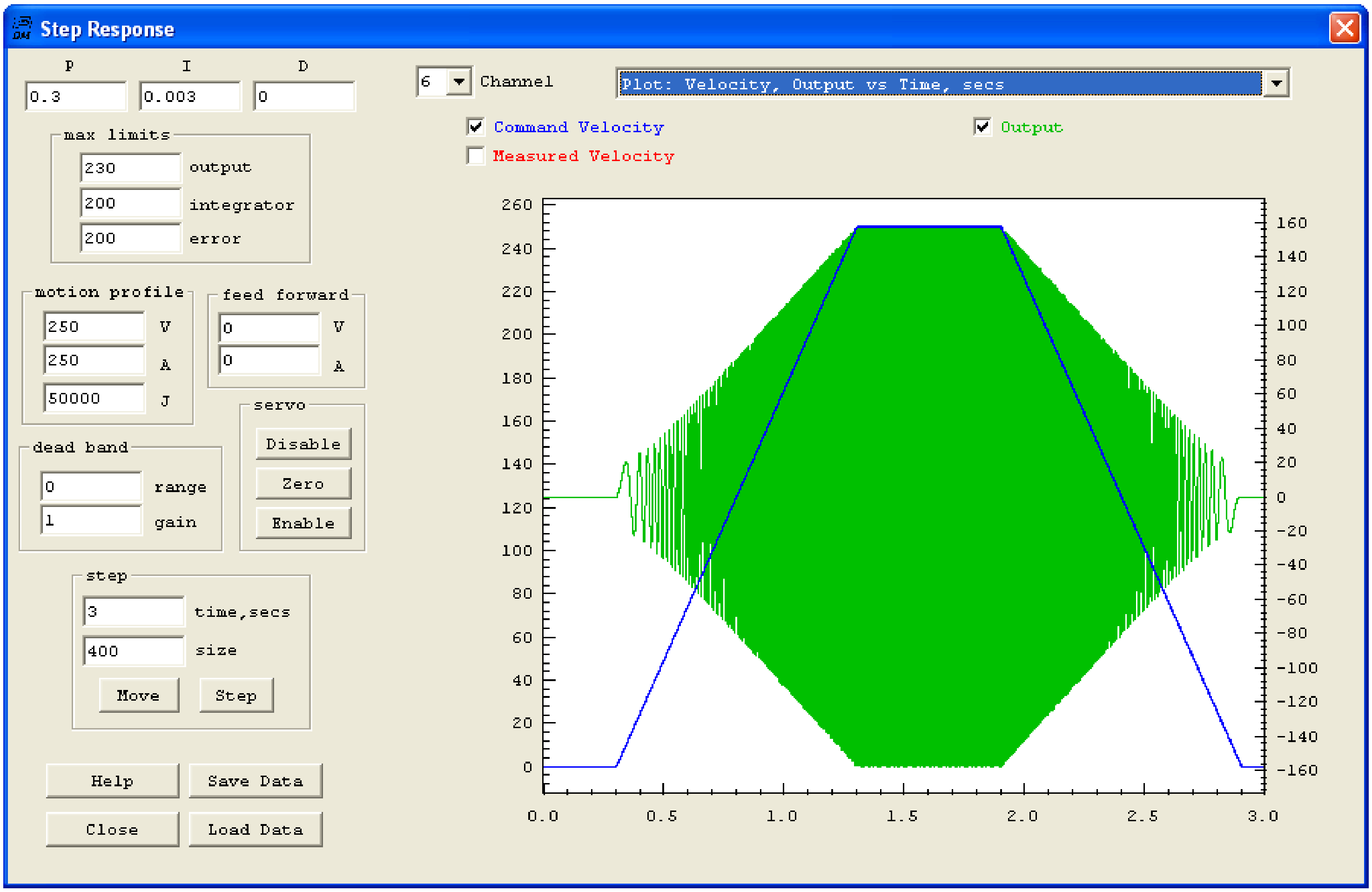
A noise from the motor is heard during about the middle half of the motion. Here the motor position and current in both stepper coils is plotted.



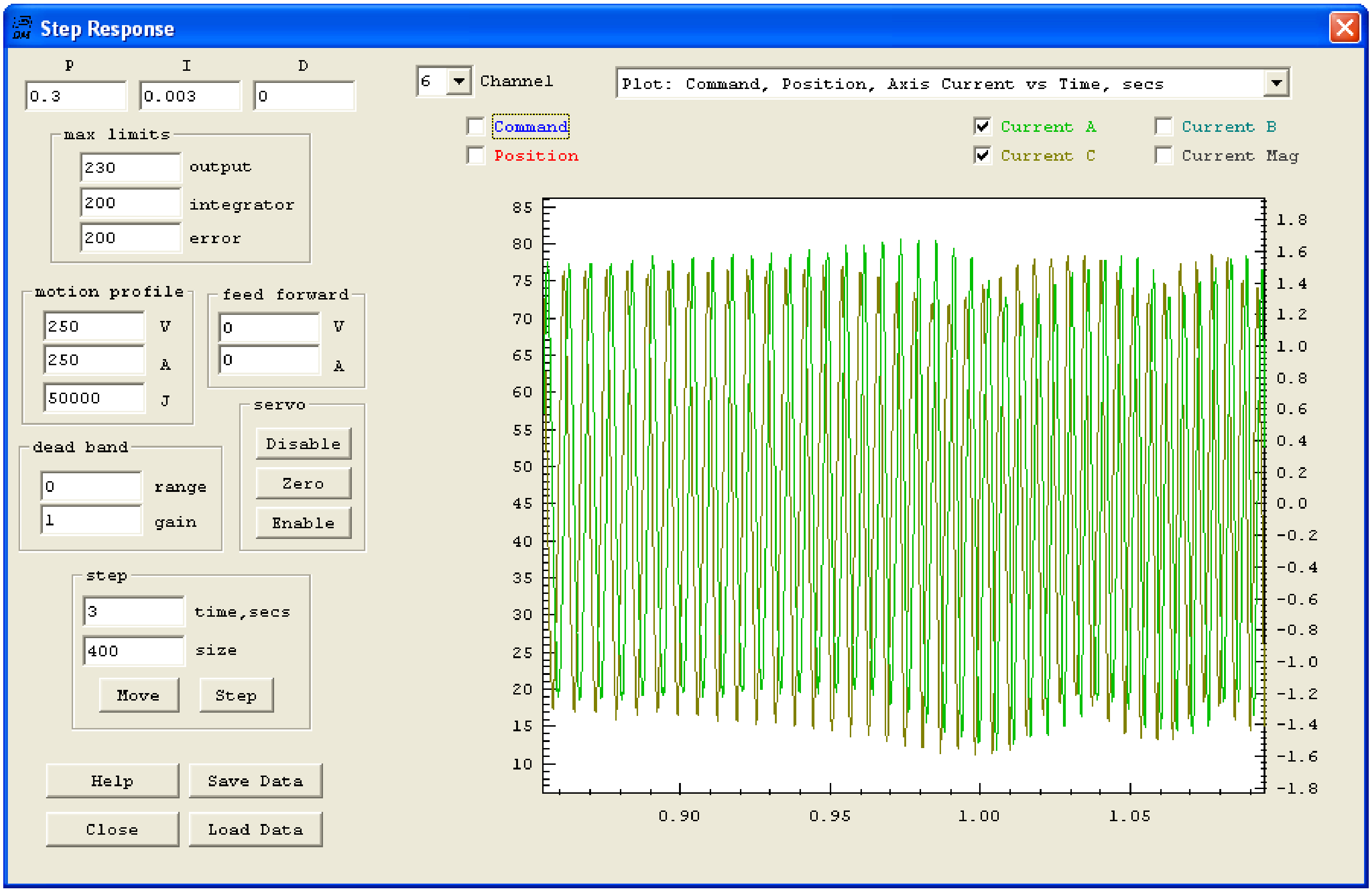
Zooming in on the beginning of the motion shows nice current waveforms 90 degrees out of phase.



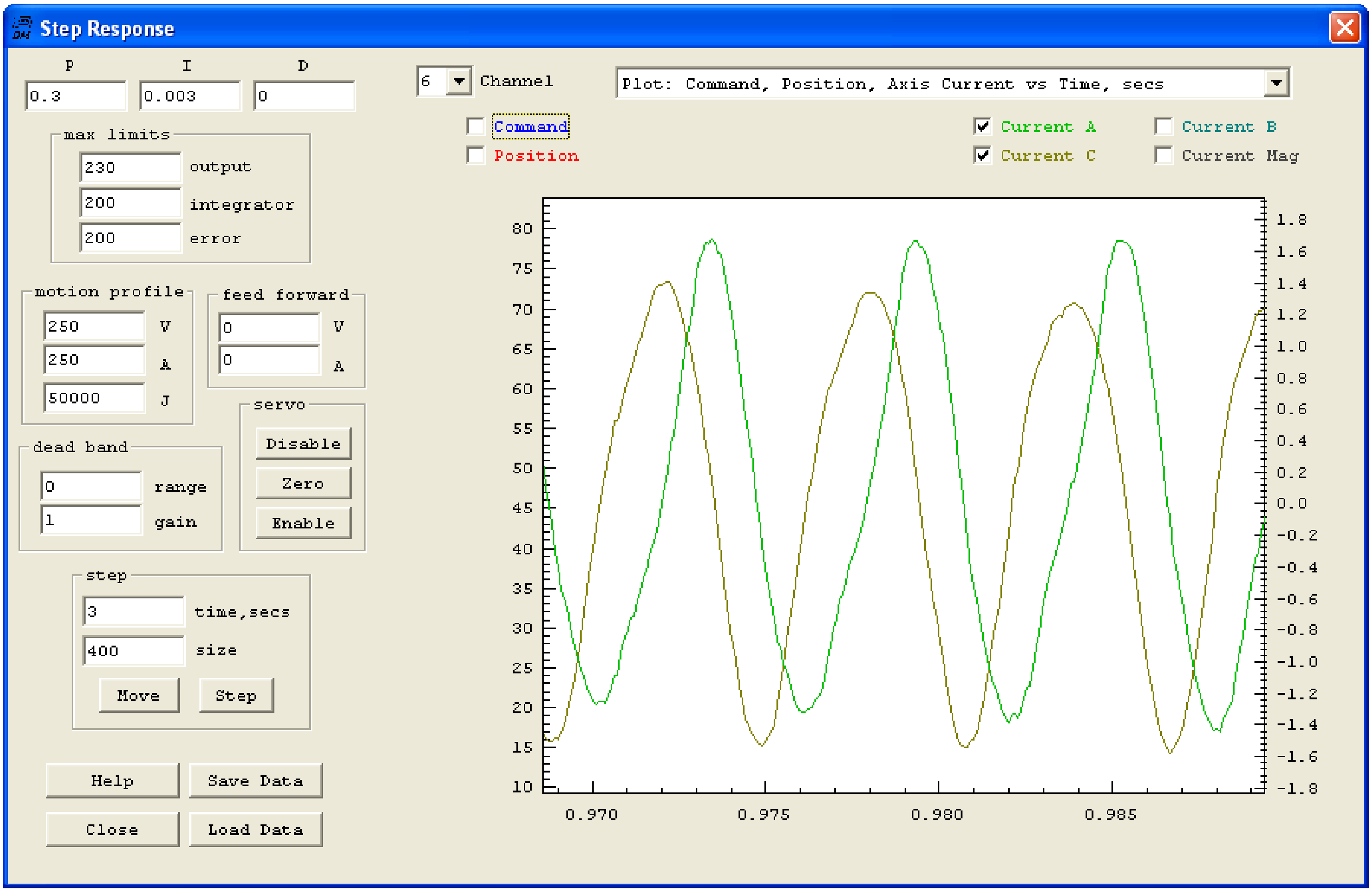
Here is a plot of the Velocity and the commanded output Voltage of one of the coils. Note how as the Velocity increases the lead compensation (set at 55) increases the amplitude. Referring back to the first current plot above we can see that the current amplitude is maintained at about 1.4 amps throughout the motion. Showing the lead compensation is set properly.



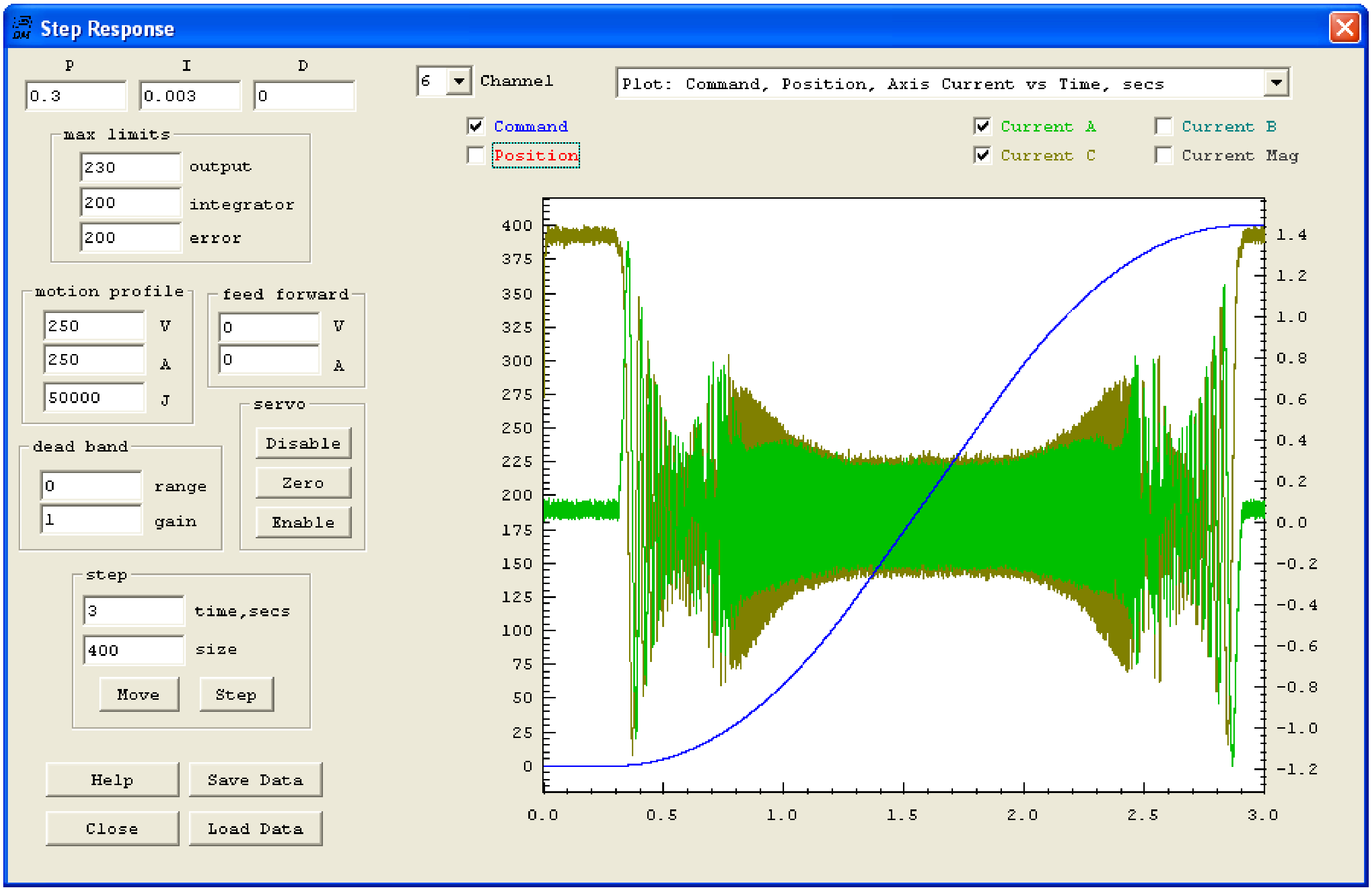
Zooming in on the current waveform at around 1 second where the motor noise seems to begin.



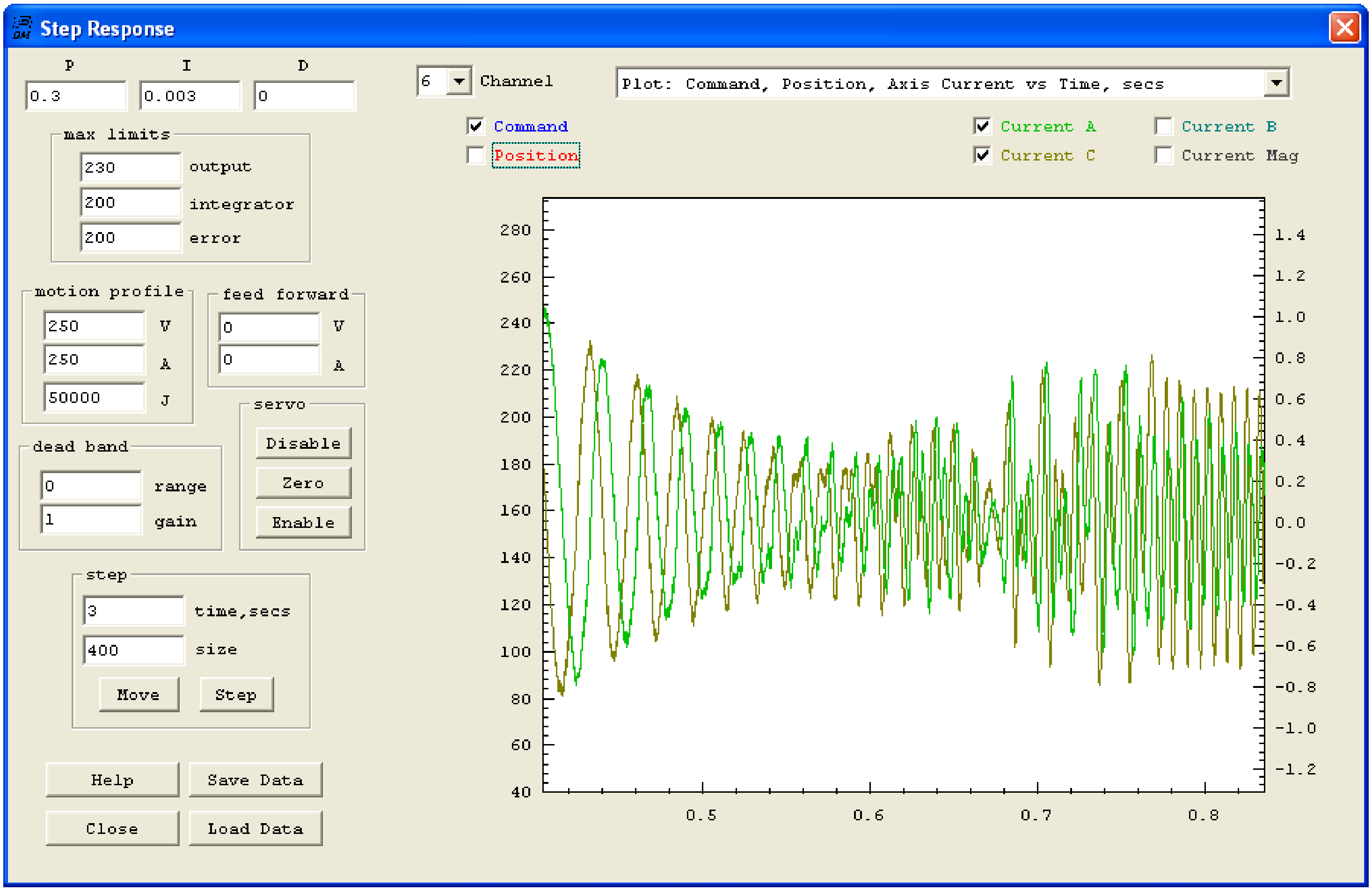
Zooming still further at 0.98 seconds where the odd behavior seems most pronounced.



Current Waveforms with no Lead - Current drops and motor immediately stalls



Zooming in on the initial stall region



KNozz Summary

3D Printer Extruder Nozzle and Part Bed Heater Interface

KNozz is the latest edition to our line-up designed to control the Nozzle Heater and Bed Heater for an Extrusion Type 3D Printer.

KNozz provides two Thermistor Inputs to allow KFLOP/Kogna to monitor the temperature of both the Extruder Nozzle and Bed Heater Platform. 12-bit ADC converter provides better than 1 degree C temperature repeatability.

Two High Current, Independent, and Optically Isolated FET switches allow controlling the Nozzle and Bed Heaters. Super low on resistance of only 2.1 milli-ohms permits continuous currents of upt to 20 Amps with no heat sinking required. Channels isolated from each other and may use separate supplies.

Safety Watchdog automatically disables heater outputs whenever temperatures are not being actively monitored.

An additional opto-isolated analog output is provided for general use.

3 LED status indicators for: Watchdog Enable, Nozzle Heat Powered and Driven, Bed Heat Powered and Driven,

Simple 16-pin ribbon connection to KFLOP JP4 provides logic power and data communication to/from KFLOP/Kogna.

CNC Application Note: KNozz may also be used in CNC Applications to provide an isolated analog Spindle Speed Output, two isolated switches for Spindle On/Off/Direction Control, and 2 analog inputs for FRO and SSO potentiometer inputs.

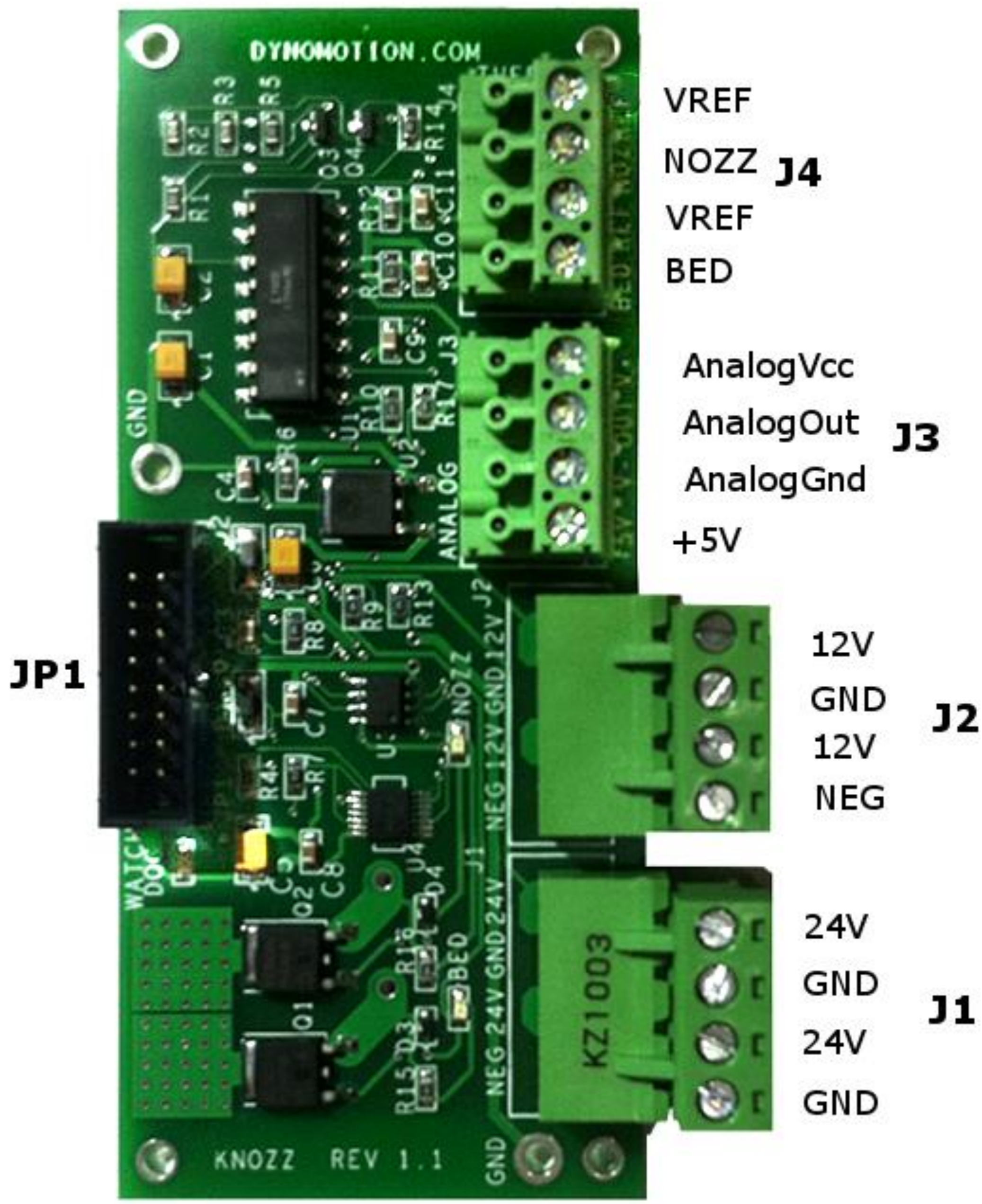
Features:

- ADC Nozzle Thermistor Input for Temperature Measurement
- ADC Bed Thermistor Input for Temperature Measurement
- 20Amp 48V Opto-Iso FET Switch for Nozzle Heater Element
- 20Amp 48V Opto-Iso FET Switch for Bed Heater Element
- Heaters Watch Dog
- Isolated PWM to Analog Converter
- 3 LED Status Indicators for Watch Dog, Bed, and Nozzle Heaters
- Thermistor Voltage Reference
- Direct Connection to KFLOP/Kogna JP6
- Screw Terminals

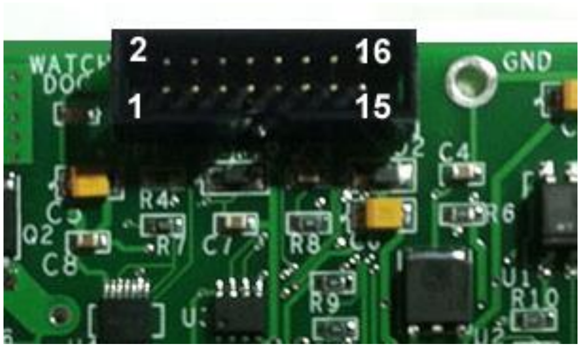
KNozz Hardware Specifications

Function	Parameter	Specification
Inputs	Number Type Resistor Divider Formula Filter Cap, Tau VRef ADC	2 Analog Inputs Themistor 100K ohm type 10K ohm 1% ADC = 10K / (10K + Rin) * 4095 0.1uF, RC = 1 millisecond 3.3V +/- 1% 12-bit
Outputs	Number Type Max Current Max Voltage On Resistance	2 FET Outputs (NPN Type) Full Opto Isolation 20 Amps 48V 2.1 milli ohms
Analog	Number Input Supply Range Output Impedance Response Tau	1 Fully Isolated Output 5-30V <1K ohm 10ms
Connectors	JP1 Aux Bus to KFLOP J1 - Bed Heat Element J2 - Nozzle Heat Element J3- Analog J4 - Thermistors/VRef	16 pin Header (IDC 0.1 inch pinch) 4 pin Pluggable Term 5mm Amphenol-FCI TJ04715000J0G 4 pin Pluggable Term 5mm Amphenol-FCI TJ04715000J0G 4 pin Pluggable Term 5mm Phoenix Contact 1934887 4 pin Pluggable Term 5mm Phoenix Contact 1934887
Logic Supplies	Voltage VDD5 Max Current	+5V ±10% (supplied by KFLOP) 0.1 Amps
Environment	Operating Temperature Storage Temperature Humidity Lead	0-40° C 0-40° C 20-90% Relative Humidity, non-condensing Lead Free RoHS
Dimensions	Length Width Height	2.0 inches (50.8mm) 4.5 inches (114mm) 0.75 inches (19mm)

KNozz – Connector Pinouts



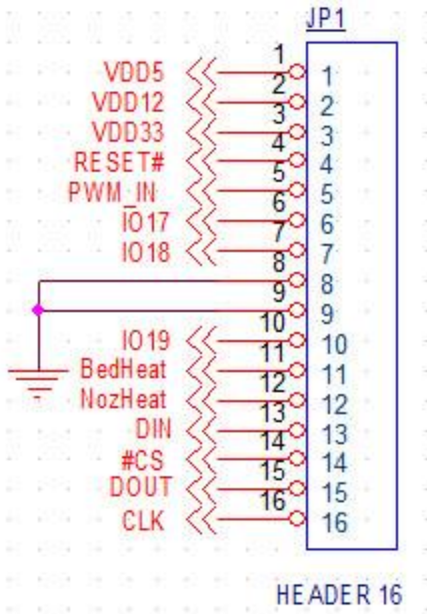
KNozz Connection to KFLOP/Kogna JP1



JP1 provides all internal signal and power connections to KFLOP/Kogna. This 16 pin ribbon connection should be as short as possible to avoid noise and crosstalk as the cable forms a high speed communication link. It can be connected to either KFLOP/Kogna JP4 or JP6. However for the Analog Output to be functional JP6 must be used which has KFLOP's/Kogna's PWM Outputs.

PWMIN is used for the isolated Analog Output. BED HEAT and NOZ Heat control the two High power FET outputs. The ADC signals are SPI signals to the dual channel ADC for reading the Thermistor Temperature readings. See KNozz C Program Examples.

Pin	KFLOP Name (JP4)	KFLOP Name (JP6)	KNozz Name
1	VDD5	VDD5	VDD5
2	VDD12	VDD12	
3	VDD33	VDD33	VDD33
4	RESET#	RESET#	
5	IO16	IO26	PWMIN
6	IO17	IO27	
7	IO18	IO28	
8	GND	GND	GND
9	GND	GND	GND
10	IO19	IO29	
11	IO20	IO30	BED HEAT
12	IO21	IO31	NOZ HEAT
13	IO22	IO32	ADC DIN
14	IO23	IO33	ADC #CS
15	IO24	IO34	ADC DOUT
16	IO25	IO35	ADC CLK



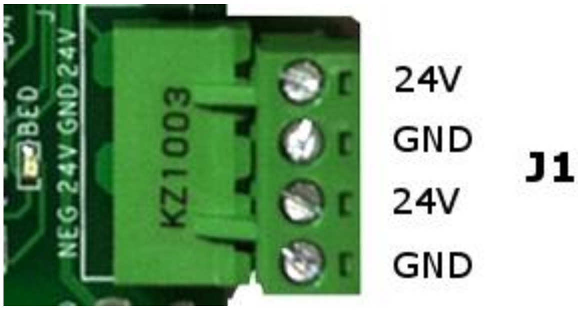
KNozz J1 and J2 High Power FET Outputs

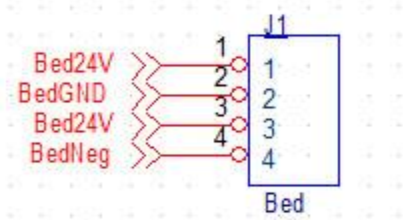
KNozz provides two independent opto isolated high current FET switch outputs. Although 3D Printer Bed Heaters are often driven with a 24V supply and Nozzle Heaters are often driven by a 12V supply any voltage from 12 to 48V can be applied and switched.

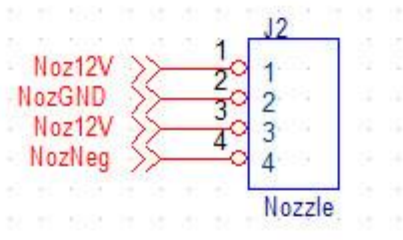
Connect the + Supply Voltage to Bed24V (or Noz12V) and the Supply GND to BedGND (or NozGND). Connect the Heater Element between Bed24V (or Noz12V) and the BedNeg (or NozNeg) terminals. The "Neg" terminals are driven the Supply GND when active and open circuit when inactive.

Both Outputs have an associated LED to indicate when Voltage is applied to the Output. The Output and LED will only be active when the WatchDog is active, the associated input is being driven, and a power supply is attached and providing power.

The BED HEAT (or NOZ HEAT) inputs are active high (they must be driven > 2.0V to enable the FET Outputs).







KNozz J3 Analog Output

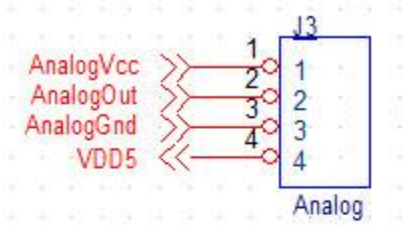
KNozz provides one opto isolated Analog output. A positive isolated supply voltage should be applied across AnalogVcc and AnalogGnd. The AnalogOut voltage will then be a function of the PWM duty cycle. Note the Output is normally isolated from KFLOP GND so make sure to use the AnalogGnd as the voltage reference.

The PWM input pin is Pin5 of JP1. KFLOP JP6 Aux1 connector has PWM0 output on this pin. The AnalogOut function will only work when KNozz is connected to KFLOP JP6 Aux1.

The PWM to Analog Circuit used is the same as for KStep. See the KStep documentation for more information.

See the Spindle_S_KNozz.c example for how to interface KMotionCNC's S command to the Analog Output.

A Terminal with +5V connected to KFLOP +5V is also provided.



KNozz J4 Thermistor Analog Inputs

KNozz provides two 12-bit analog inputs normally used for Nozzle and Bed Temperature measurement. A C Program running in KFLOP/Kogna is normally used to bit-bang SPI to read the ADC values, determine the temperatures, and control the Heaters. See the C Program KNozz Examples.

A single 3.3V (+/-1%) Reference is also provided that is also used for the Internal ADC Converter. Normally Thermistors used with 3D Printers change resistance from ~100K ohms at room temperature to ~5K ohms at build temperature. The external Thermistor should be connected between the VREF terminal and the NOZZ (or BED) Terminals.

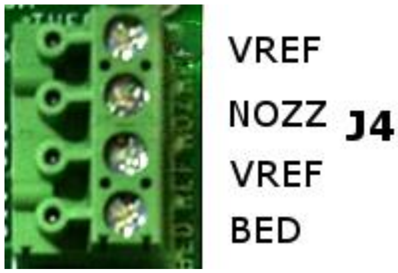
A Thermistor and internal 10K ohm (+/- 1%) resistor form a Voltage divider. See the circuit below.

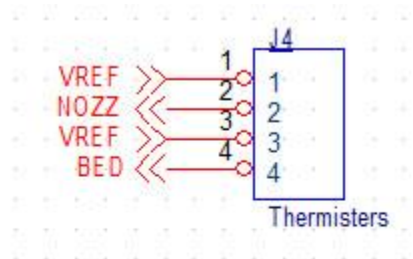
The relationship between input Voltage to ADC reading in counts is:

ADC = 4095 x Vin / VREF

The relationship between resistance R and ADC reading in counts is:

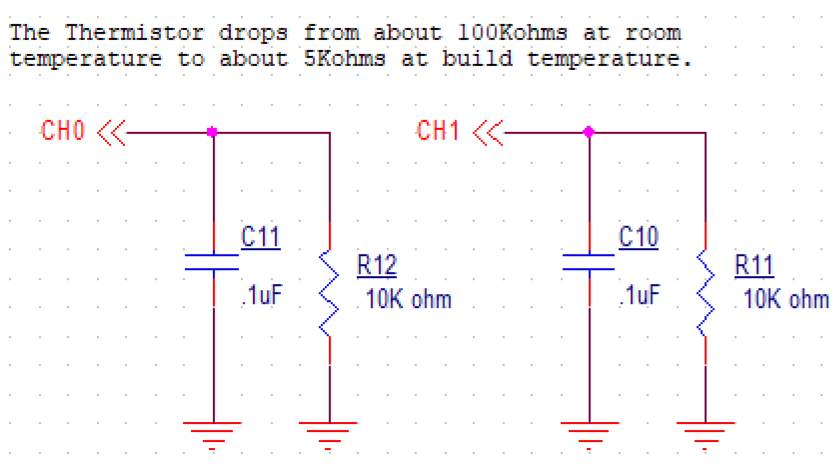
ADC = 4095 x 10K / (10K + R)





Each Thermistor input circuit includes a filter capacitor (0.1uF) that will slow response and reduce noise. The response time will change depending how the input is driven but at high resistance the input response will be:

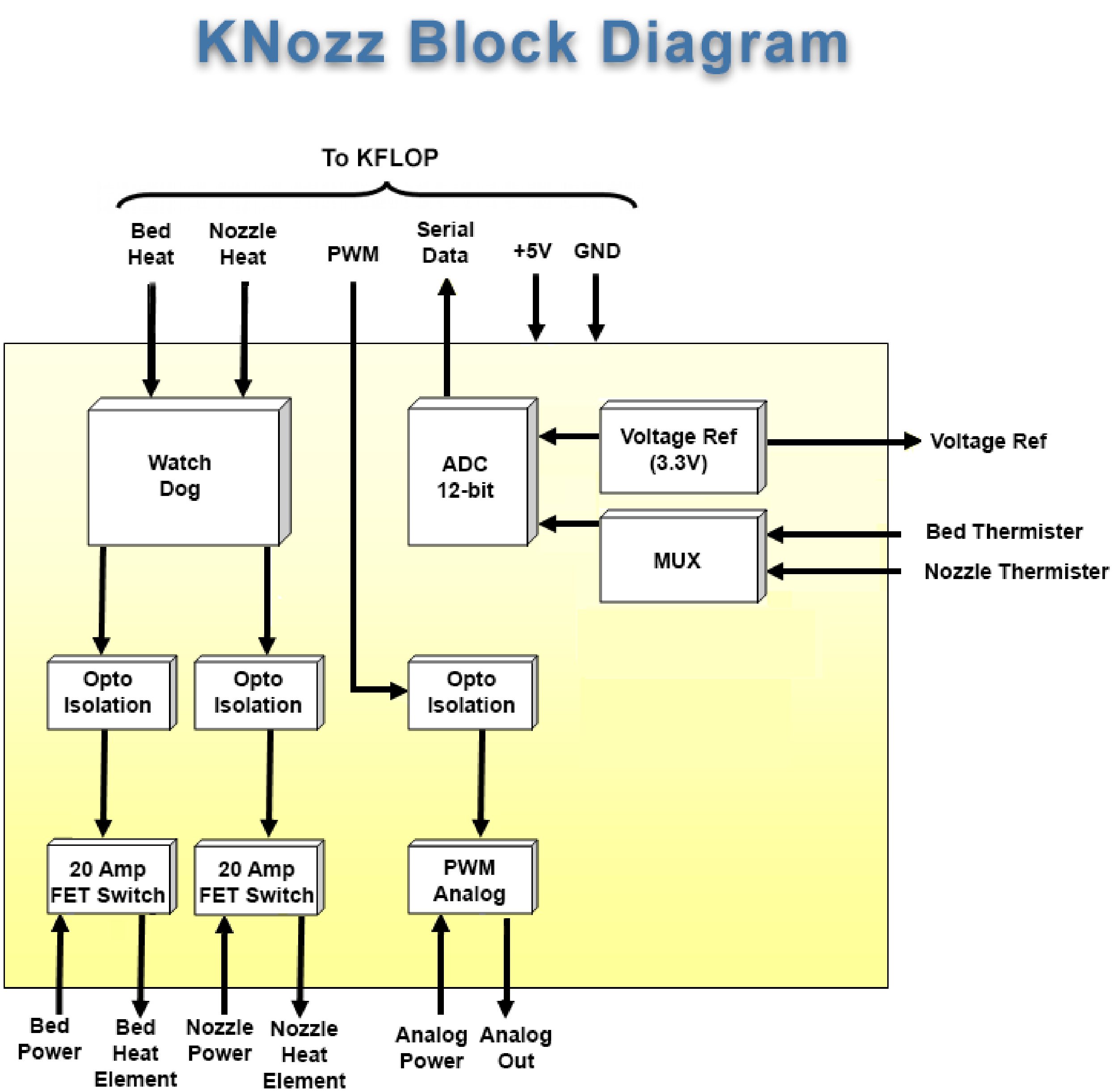
Tau = RC = 10K x 0.1uF = 1ms



WatchDog

KNozz includes a WatchDog Circuit to help avoid thermal runaway of the Heaters if the FET Outputs are somehow turned on while the temperatures are not being actively monitored. To read the temperatures the ADC CLK input must be toggled. If the ADC Clock stops togging, then the WatchDog Circuit trips and disables both FET Outputs after approximately 3ms. The WatchDog LED illuminates when the ADC Clock is toggling and the Outputs are enabled to be turned on.

KNozz Block Diagram



KNozz Board Layout

Configuring Step and Direction Outputs for KFLOP

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 RPM / 60 Sec/Min * 200 steps/rev * 128 µsteps/full step = 1.28 MHz µsteps/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.

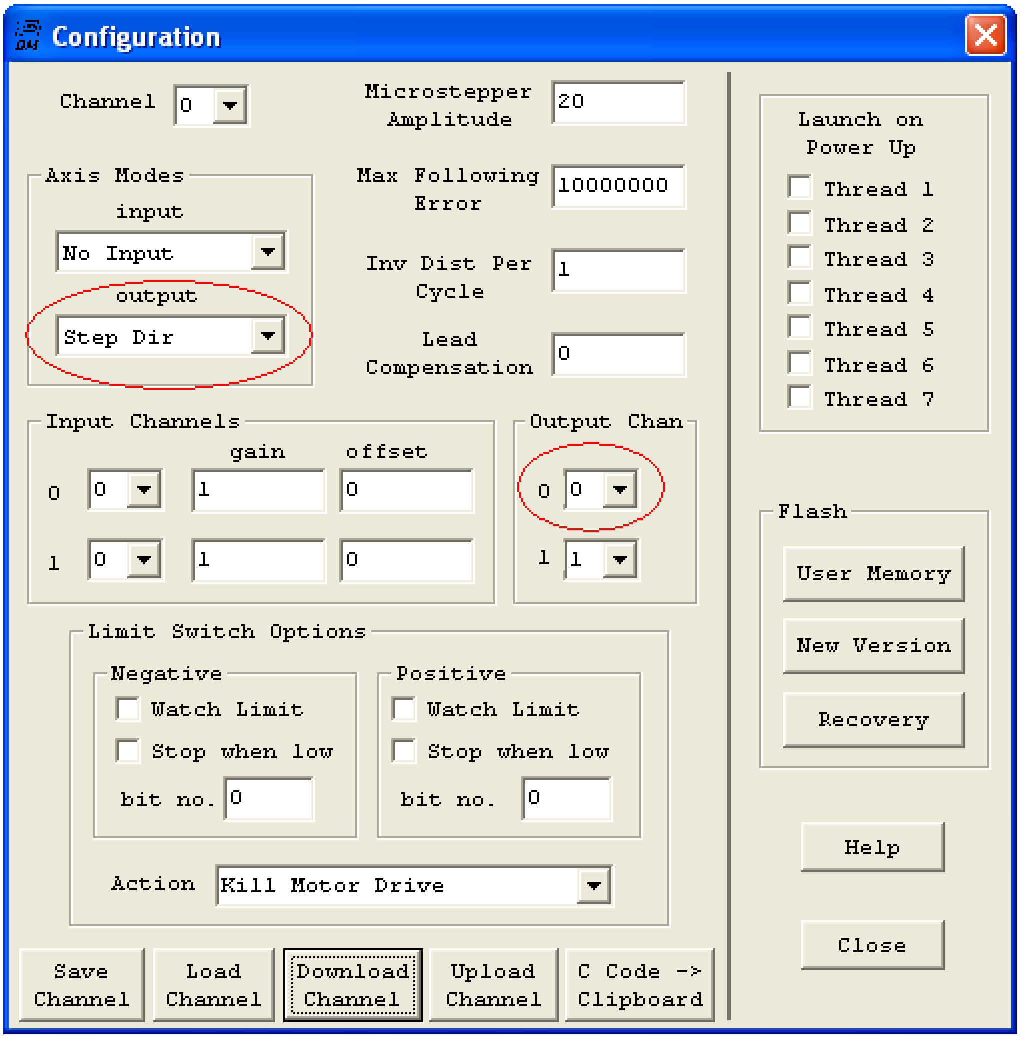
If an axis channel is selected as a Step and Direction axis, the corresponding 2 output pins will be automatically configured as outputs and they may not be used as general purpose IO.

For KFLOP see the Table below:

	Mux = 0		Mux = 1	
Signal	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.

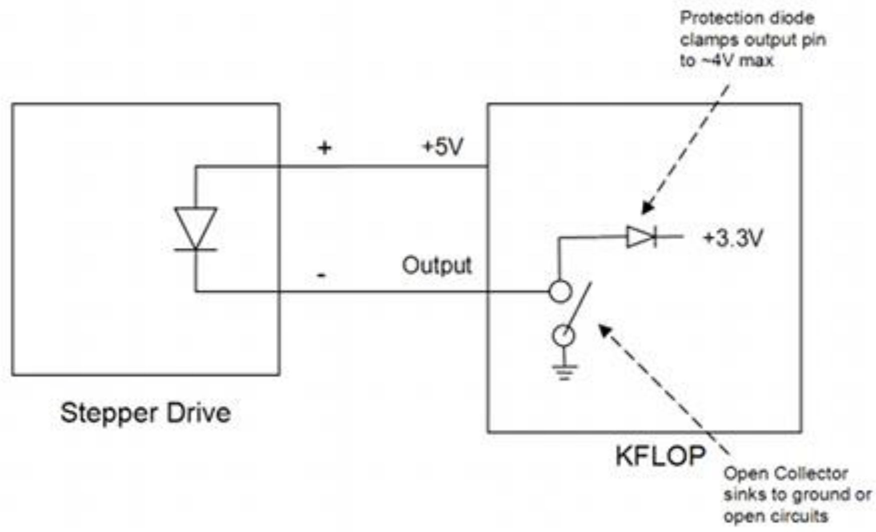


Valid output channel settings 0-31 are allowed for KFLOP. Although only 8 Step and Direction generators are available for KFLOP by adding 8 and/or 16 to the channel number the mode of the Step/Dir Generator can be changed.

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 LVTTL 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/sec^2, and micro-steps/sec^3 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 x 2us) = 250KHz.

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:

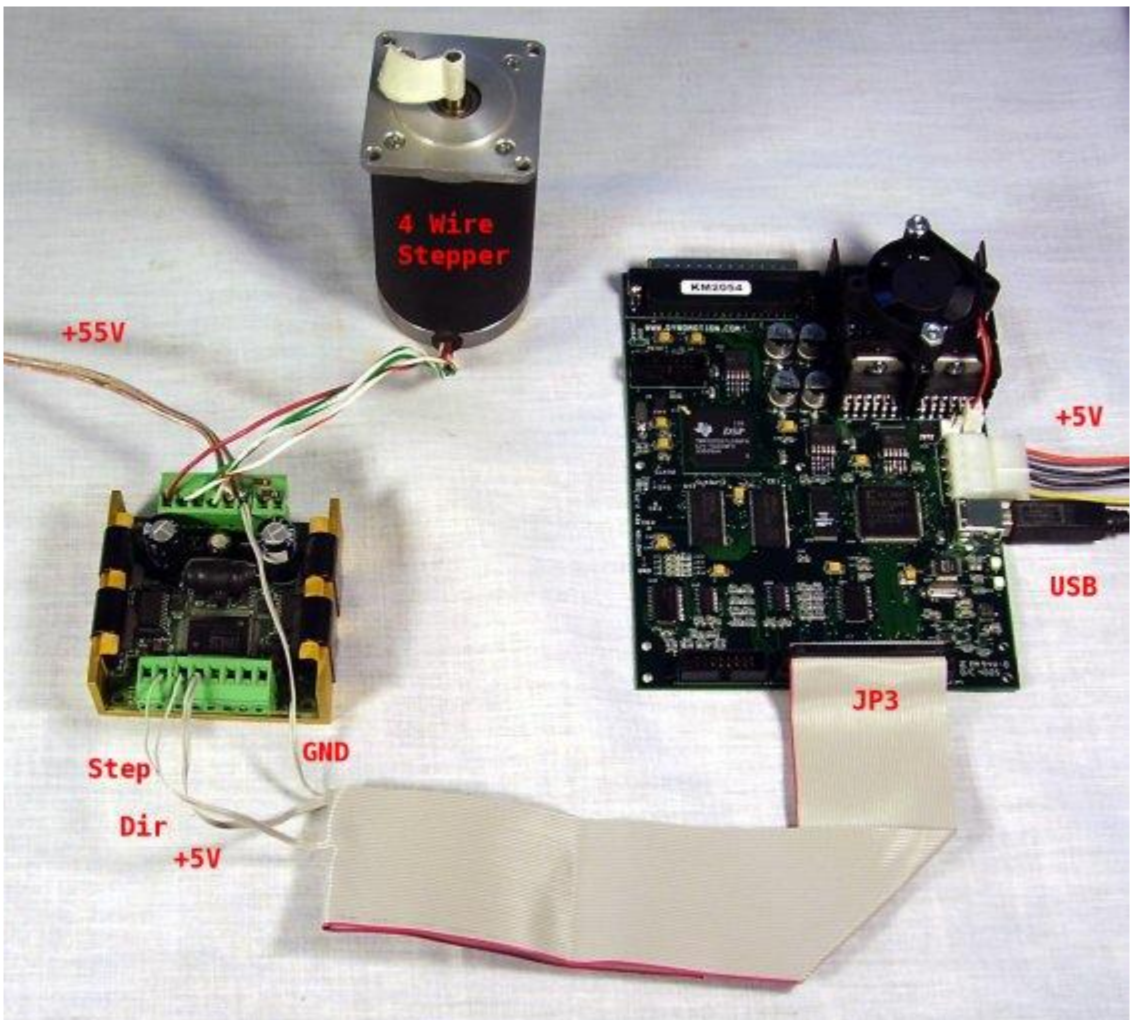
```
FPGA(STEP_PULSE_LENGTH_ADD)=32; // set the pulse time to ~ 2us
```

FPGA(STEP_PULSE_LENGTH_ADD)=32 + 0x40; // set the pulse time to ~ 2us and multiplex to JP4 and JP6

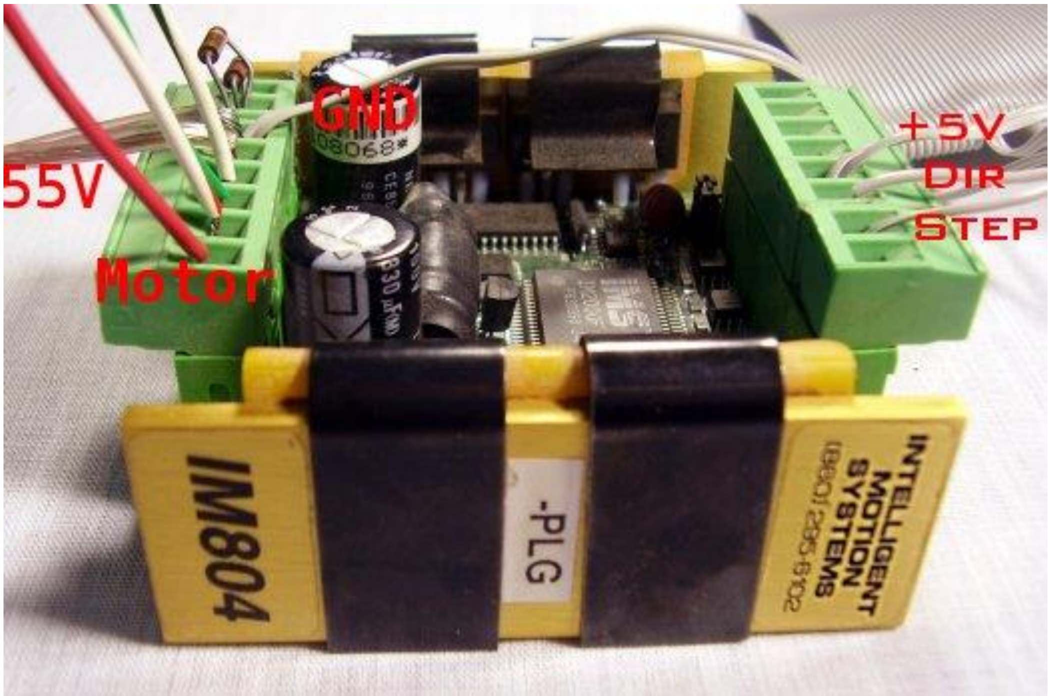
FPGA(STEP_PULSE_LENGTH_ADD)=32 + 0x80; // set the pulse time to ~ 2us and pulse the Step High

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 an IM804 Amplifier and Rapidsyn Stepper

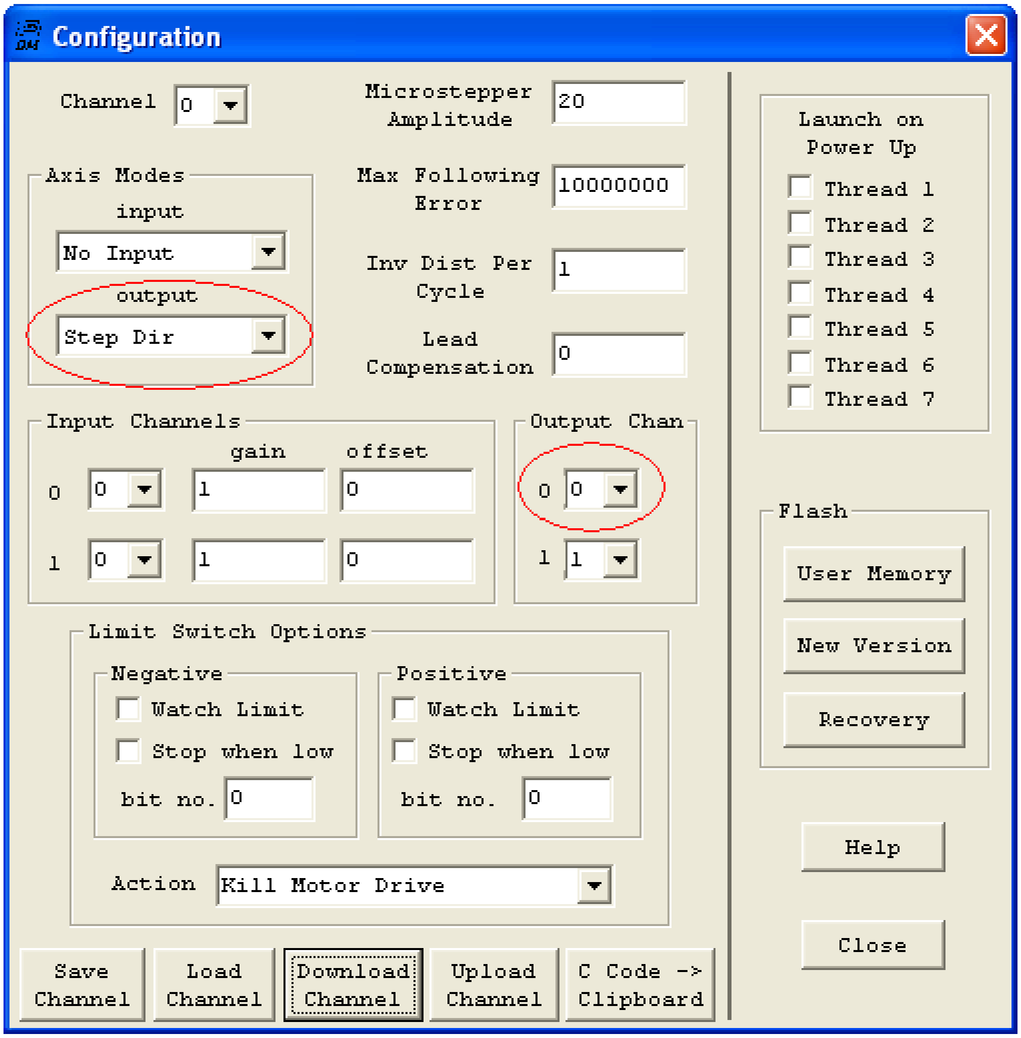


Layout - KMotion - Amplifier - Motor

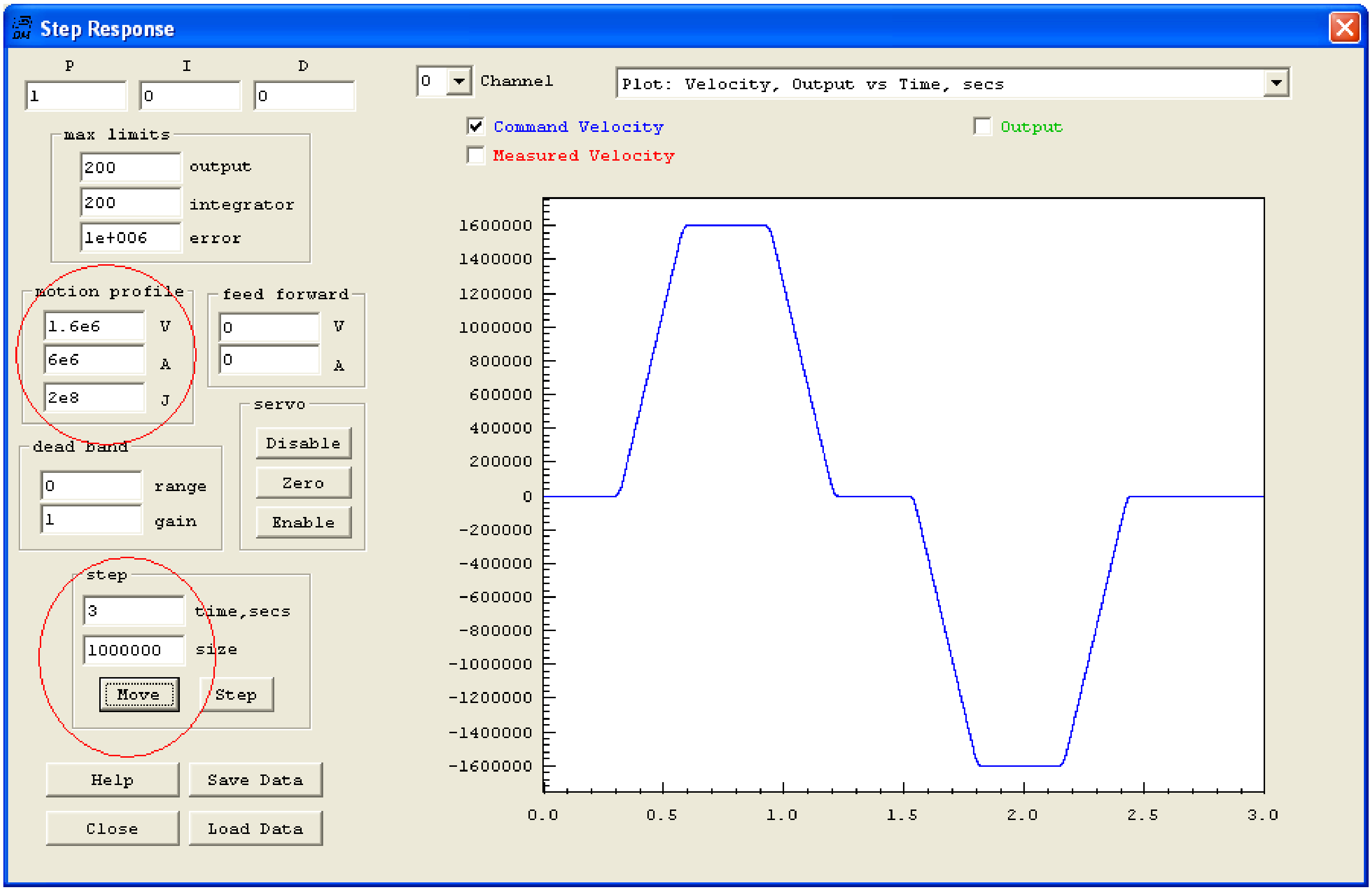


Amplifier Connections - resistors set coil currents (1500 ohms = 3A) and idle standby current (270 ohms = 0.5A). DIP Switches set to ON-OFF-OFF-ON for 128:1 micro-stepping.

Rapidsyn stepper motor. Note coil winding's centertaps are not used.



Axis channel 0 configures as "Step Dir" mode and using Step and Direction generator 0.



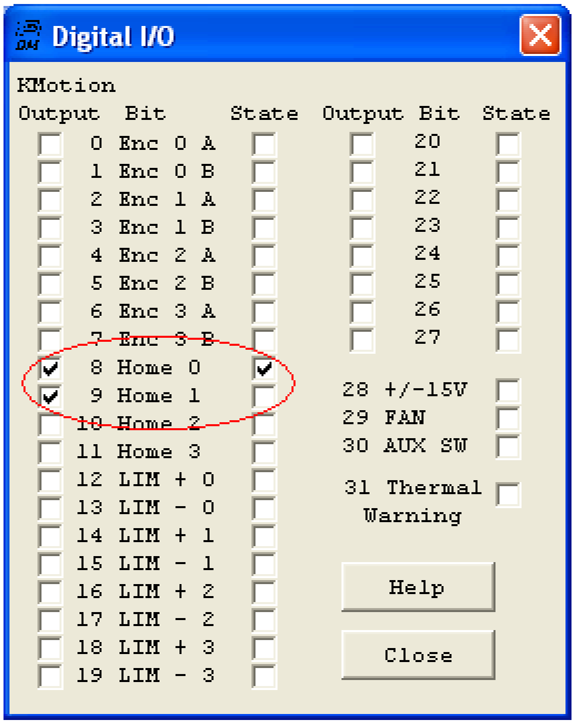
Max Velocity 1.6 x 10^6 steps/sec (1.6e6 / 128 = 12,500 sps = 3750 RPM). Note this (and most) steppers lose most all of their useful torque above several thousand full steps per second. The high speed demonstrated here is intended to demonstrate KMotion's ability to generate high pulse rates.

Max Acceleration 6 x 10^6 steps/sec^2

Max Jerk 2 x 10^8 steps/sec^3

1 x 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

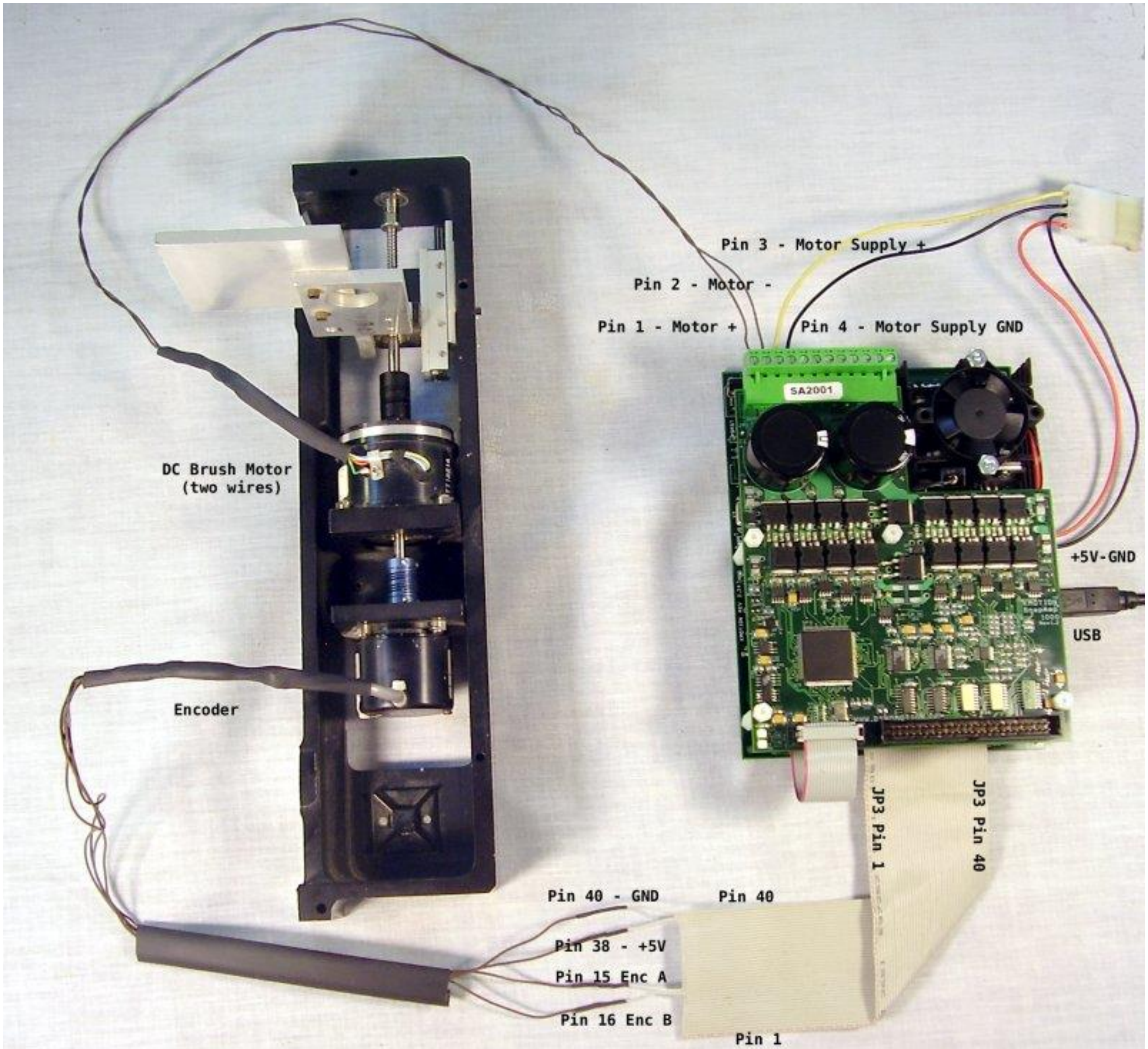
This example will proceed through the following steps

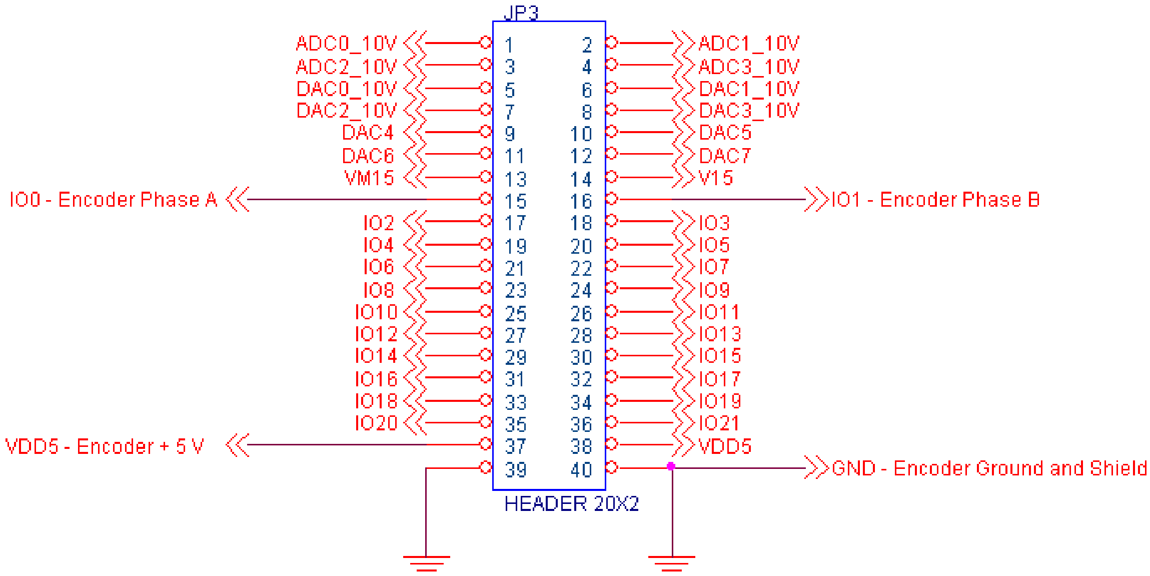
1. **Wiring the Motor, encoder, and Power**
2. **Configuring the Software**
3. **Testing the encoder**
4. **Testing the Motor/Amplifier**
5. **Closing the loop**
6. **Tuning the Servo**
7. **Executing a simple C Program Motion Sequence**

Here is a video overview.

Wiring the Motor, encoder, and Power

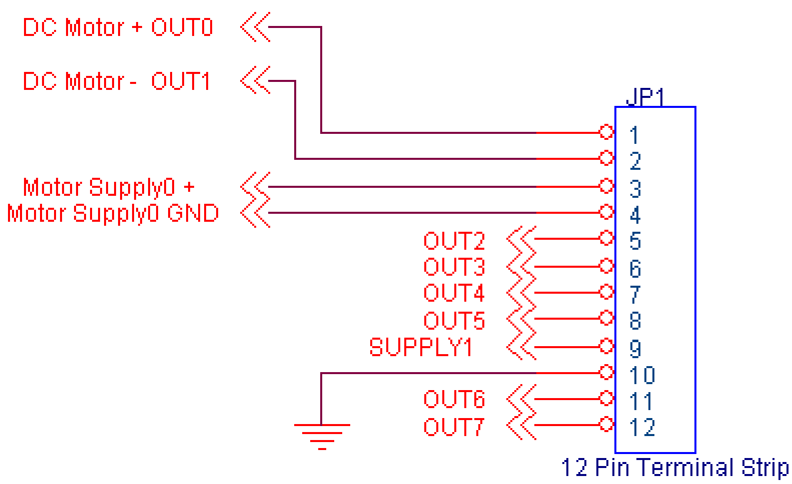
This example configuration shows a single dc brush motor connected to the first of 4 available full bridge drivers on a single SnapAmp 1000. A single-ended encoder is connected to KMotion's JP3 connector as encoder input #0 for Servo feedback. (The KMotion board is hidden underneath the SnapAmp in the 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.


Pin 3 - Motor Supply +
Pin 2 - Motor -
Pin 1 - Motor +
Pin 4 - Motor Supply GND
SA2001
DC Brush Motor
(two wires)
+5V-GND
USB
Encoder
JP3 Pin 1
JP3 Pin 40
Pin 40 - GND
Pin 40
Pin 38 - +5V
Pin 15 Enc A
Pin 16 Enc B
Pin 1



Encoder Channel 0 Wiring Diagram

KMotion to Single Ended Encoder



SnapAmp Wiring Diagram

PWM Channel 8 to DC Brush Motor

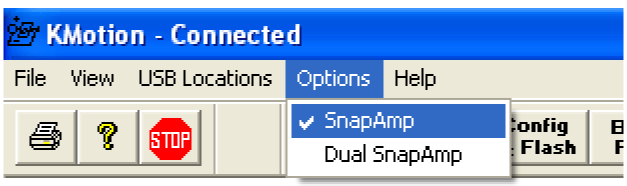
Configuring the Software

After the motor and encoder connections have been made, the software is configured and the encoder feedback and motor is tested.

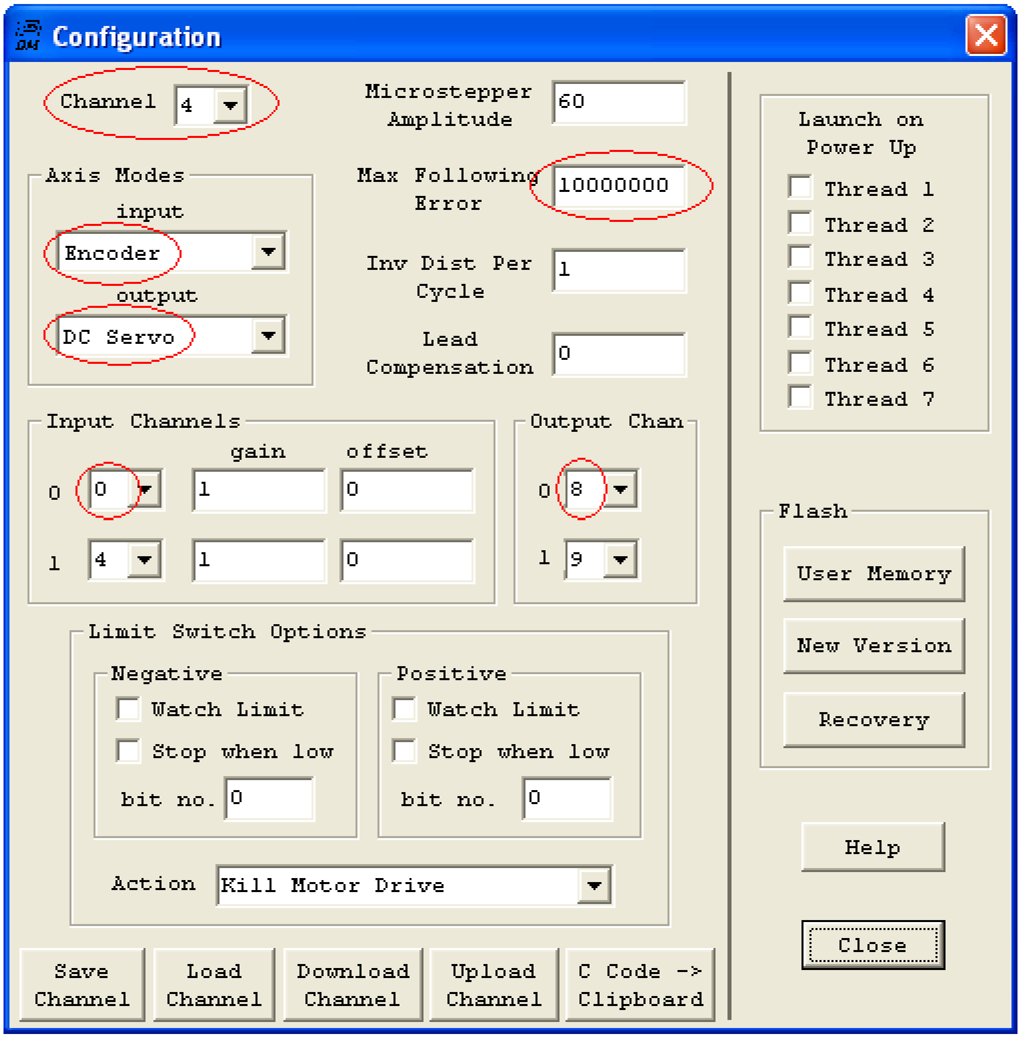
Although any of KMotion's 8 axes could be used, axis 4 will be configured since axes 0-3 are normally used for KMotion's on-board amplifiers.

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:

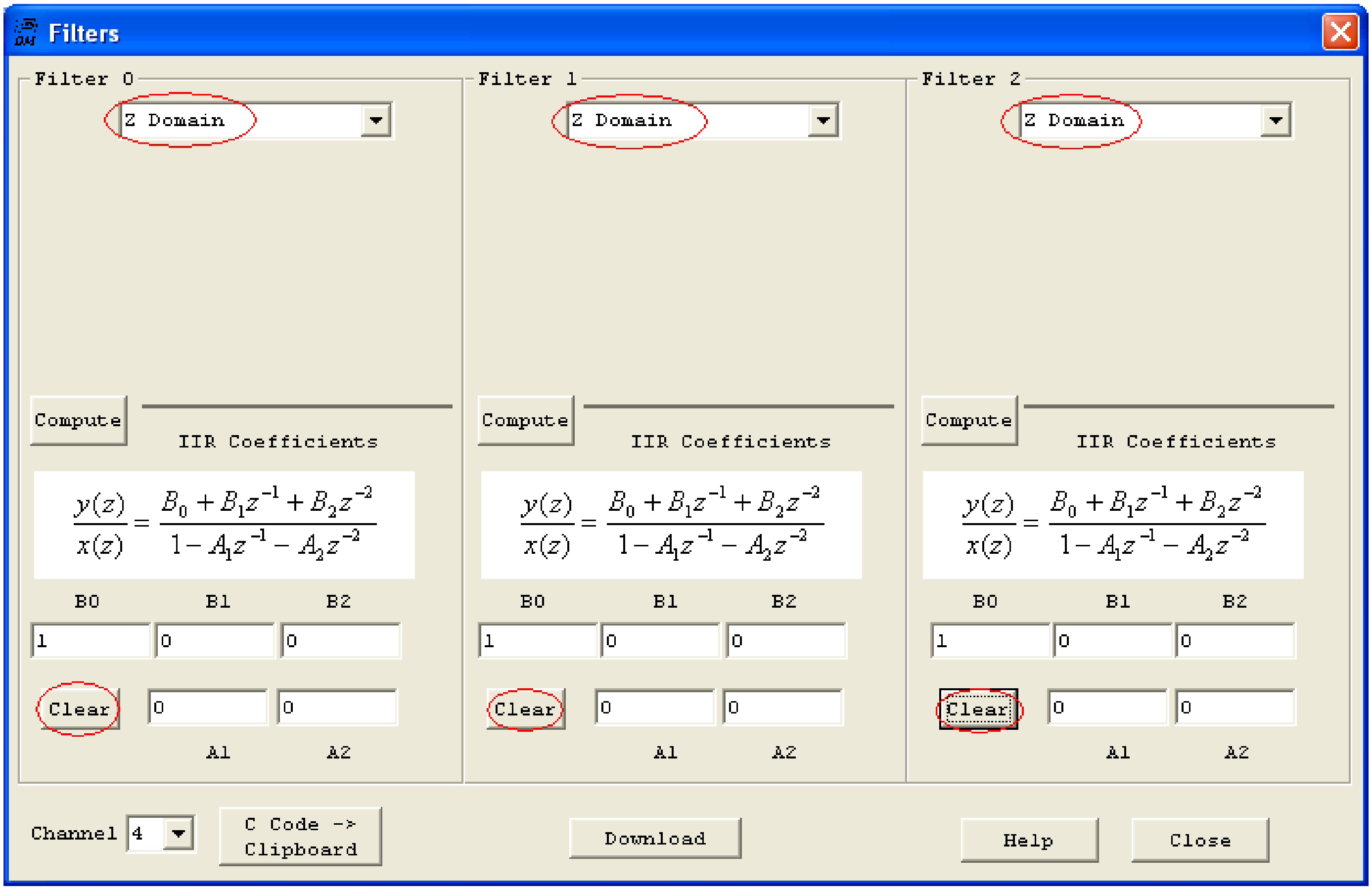
Execute the KMotion executive program and set the Options for a single SnapAmp. See below:



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.

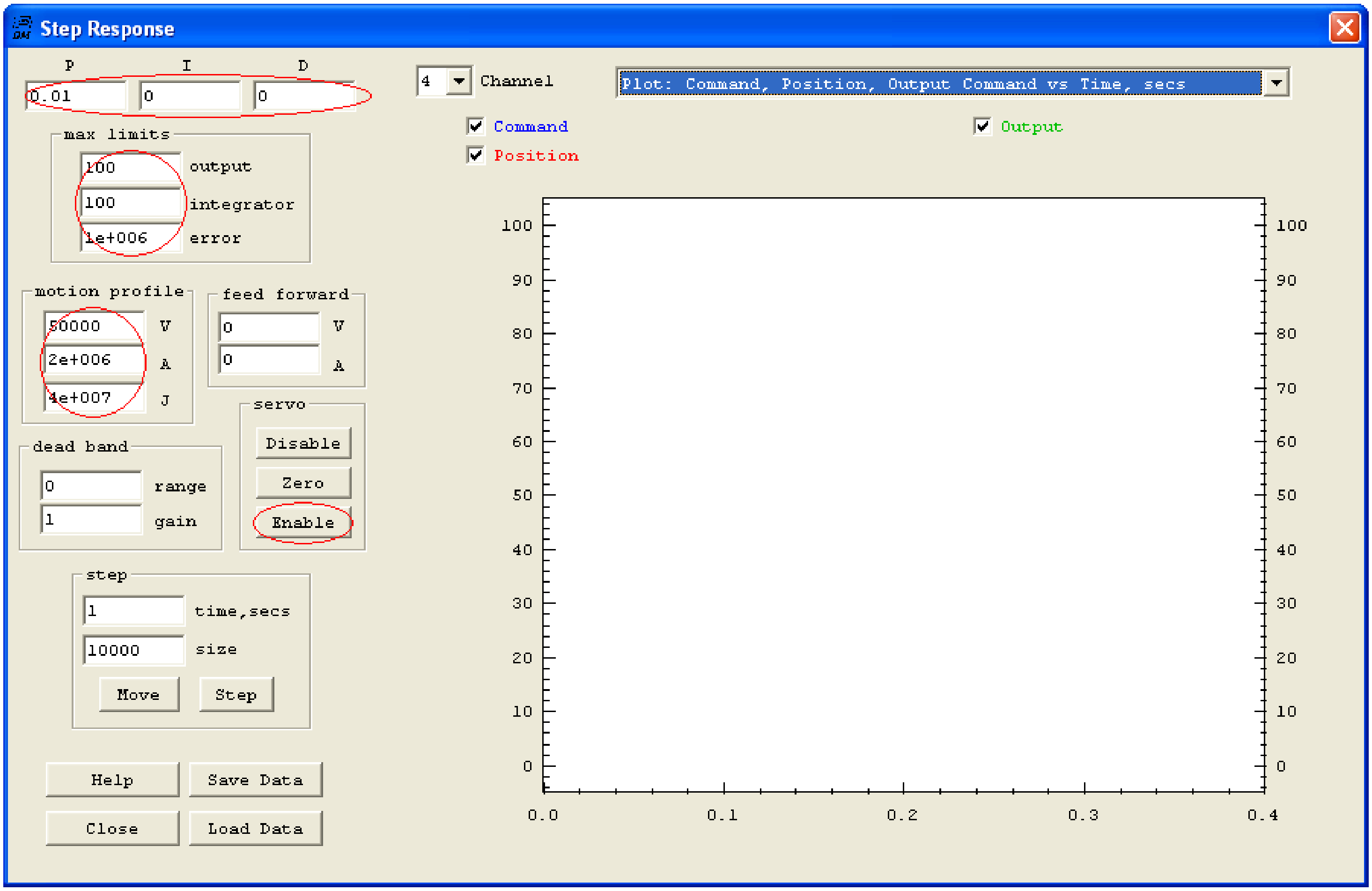

Configuration
Channel 4
Microstepper Amplitude 60
Axis Modes
input
Encoder
output
DC Servo
Max Following Error 10000000
Inv Dist Per Cycle 1
Lead Compensation 0
Launch on Power Up
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7
Input Channels
gain
offset
0 0 1 0
1 4 1 0
Output Chan
0 8
1 9
Flash
User Memory
New Version
Recovery
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
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.



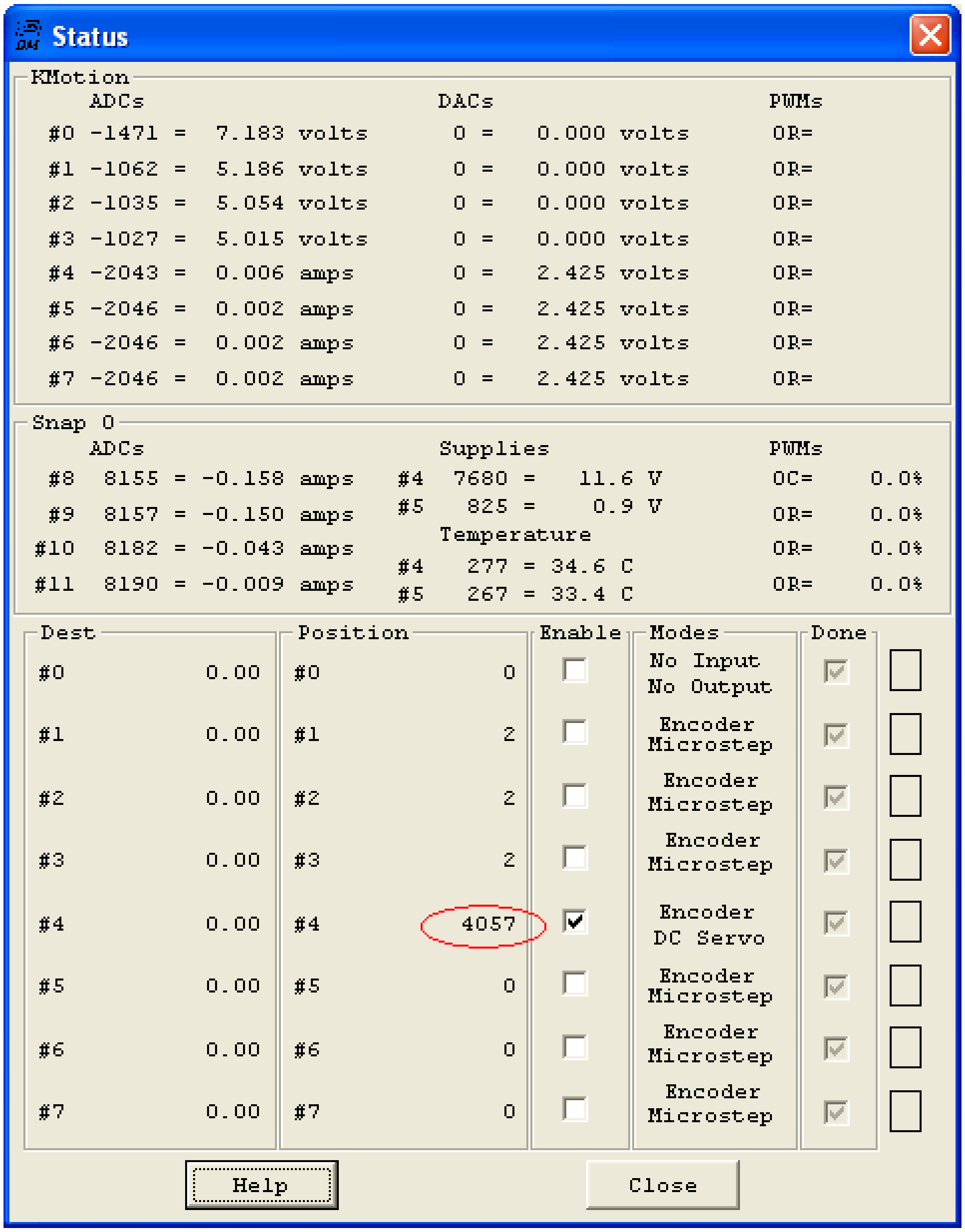
The Step Screen servo loop (PID) parameters are set to only very low proportional gain. The max output is set to a small value that should still allow some motor motion. SnapAmp DC Servo mode outputs current commands in the range of -1000 to +1000 which corresponds to approximately -30Amps to +30 Amps. 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

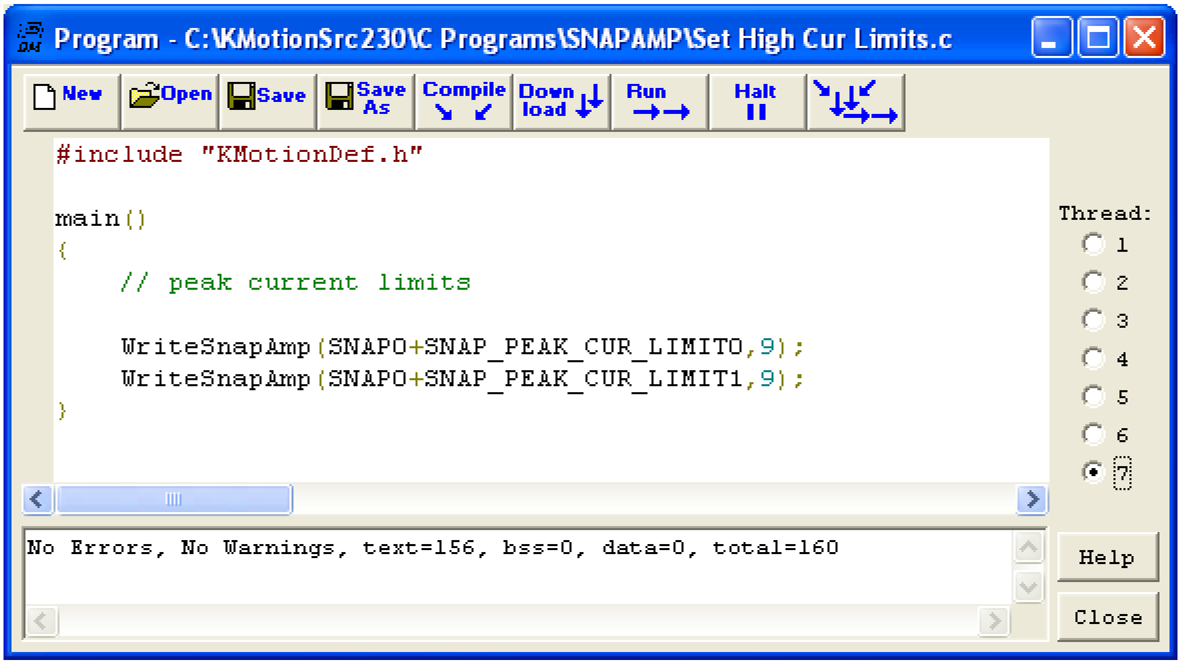
We can now rotate the motor/encoder shaft by hand and should observe counts on Axis Channel 4 on the Analog Status Screen. Note if one of the other axis's configurations also is defined as input Encoder channel 0 then that axis may count as well. The default configuration for axis 0 is Encoder input 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.



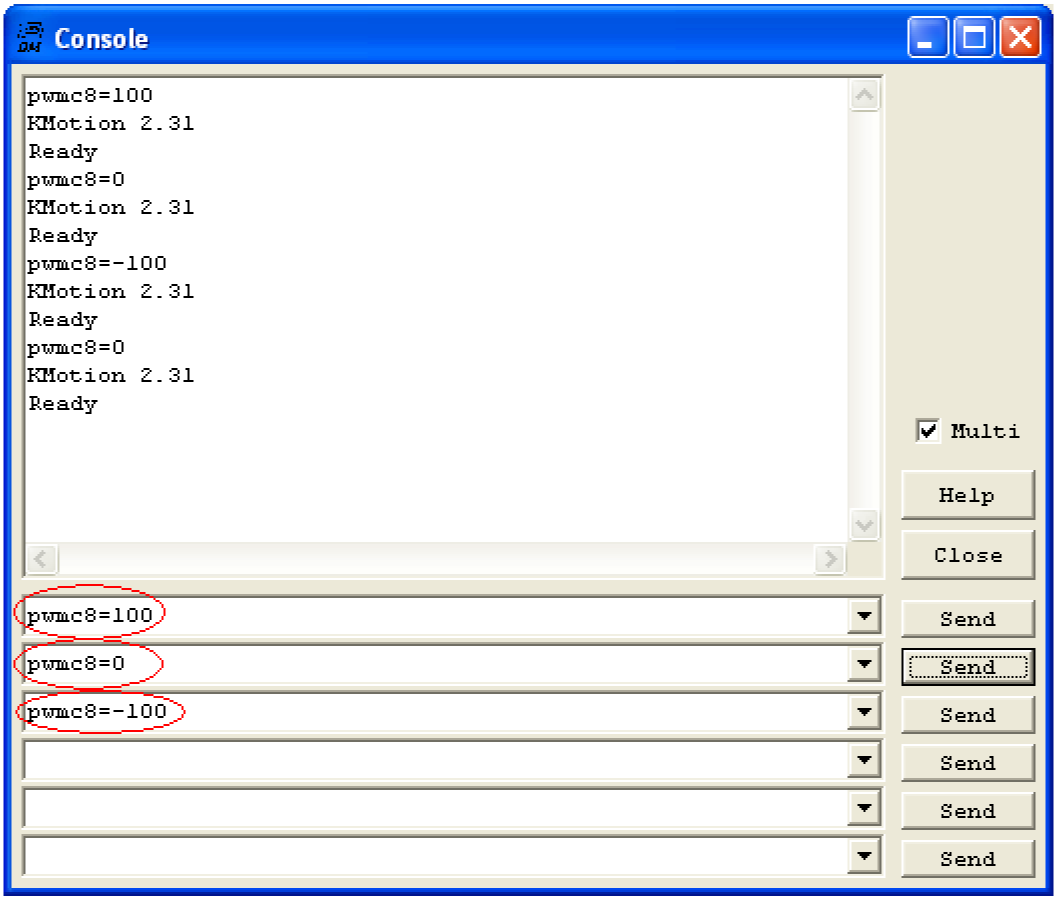
Testing the Motor Amplifier

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 PWMC 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.



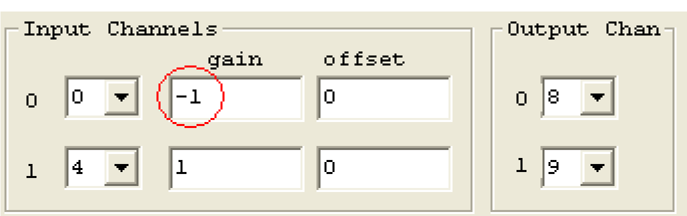
Setup the Console Screen as shown below. Sending the commands should drive the motor in one direction, then no motor drive, then drive in the other direction respectively.



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).

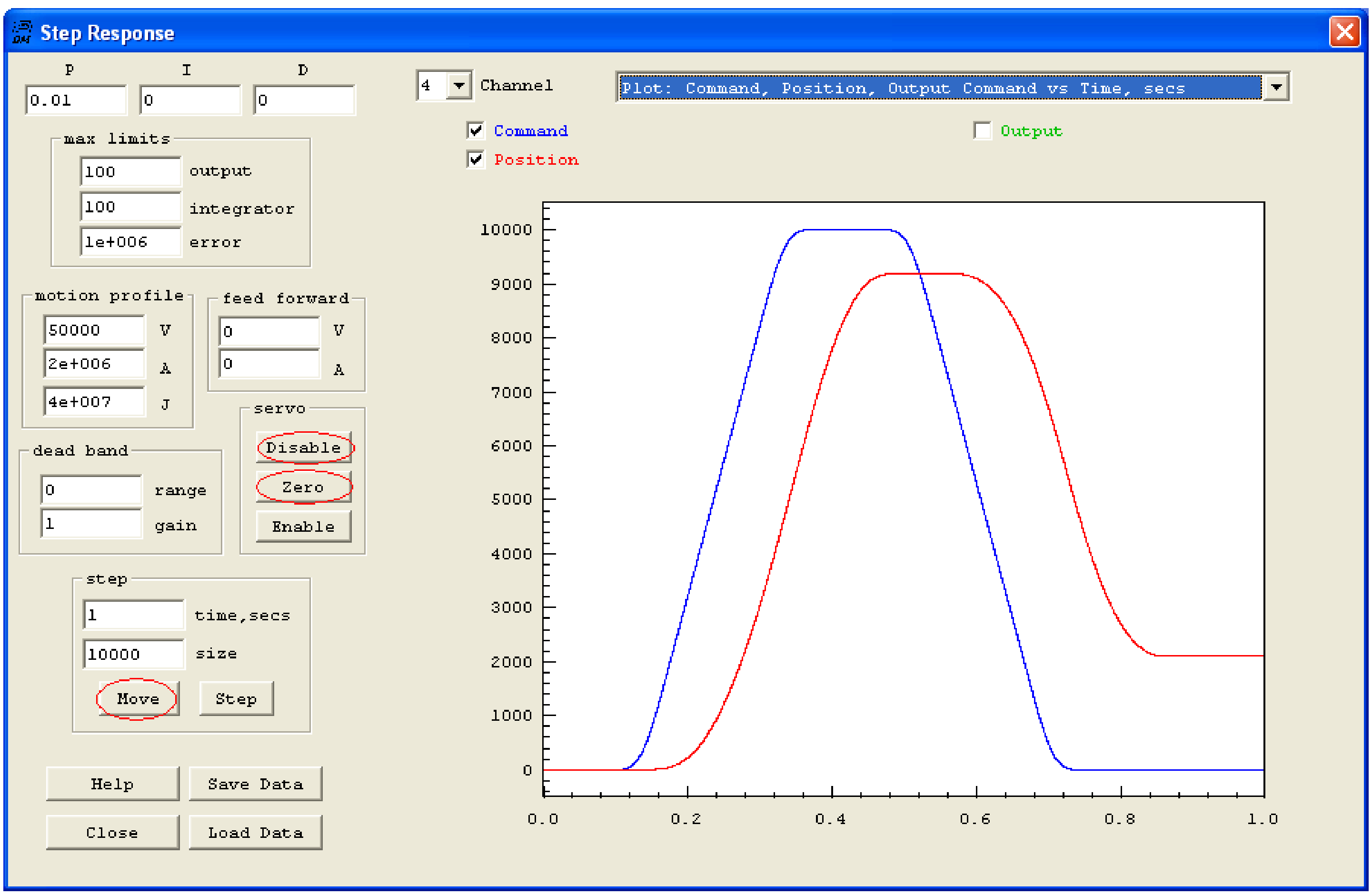


If the servo has proper feedback and responds as a weak servo, as the motor/encoder shaft is turned the servo will cause torque to restore the position back to where it was. With the example proportional gain of 0.01, turning the sharft 1 turn should generate an error of 4000 counts. 4000 counts multiplied by a gain of 0.01 = 40 PWMC counts or 40 / 1000 * 30Amps = 1.2 Amps.

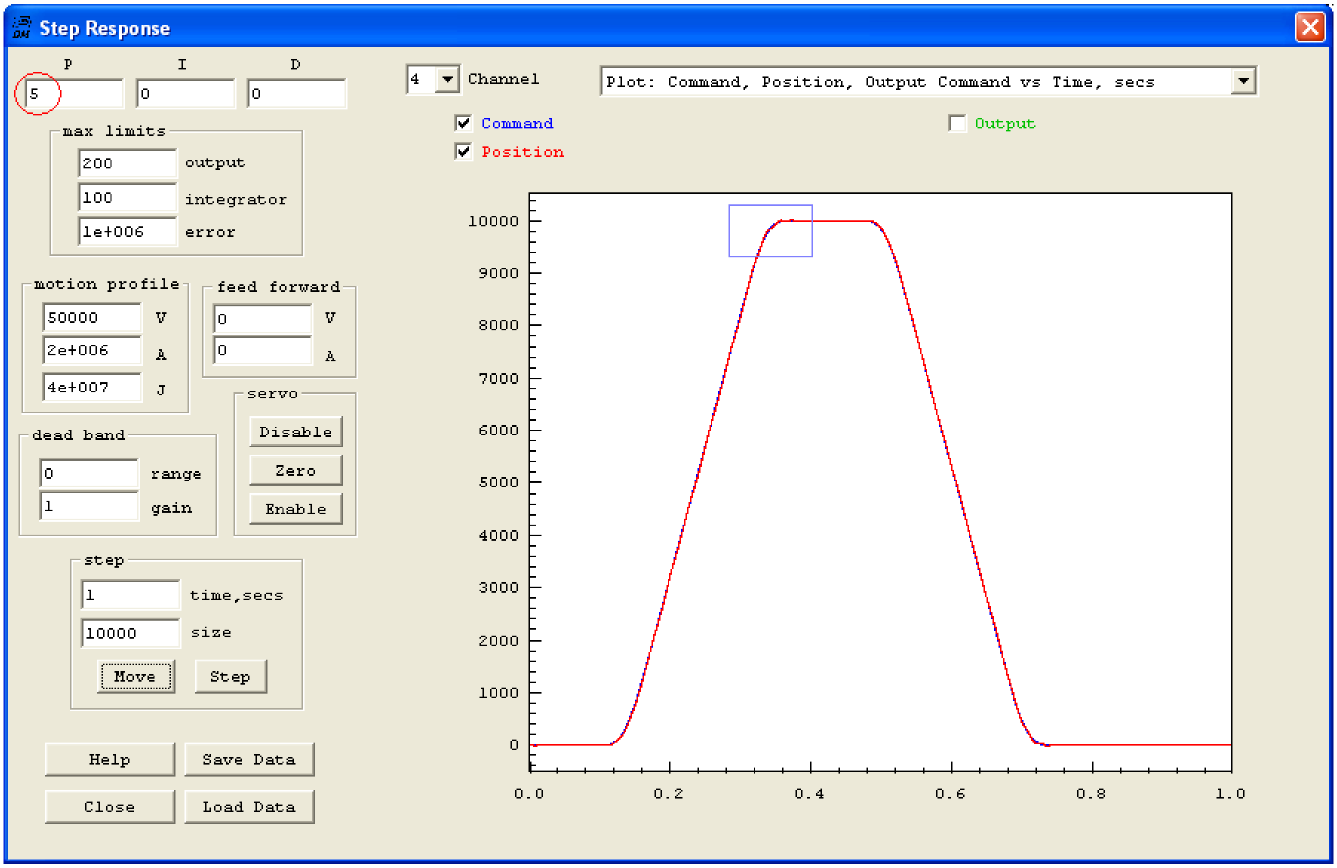
Tuning the Servo

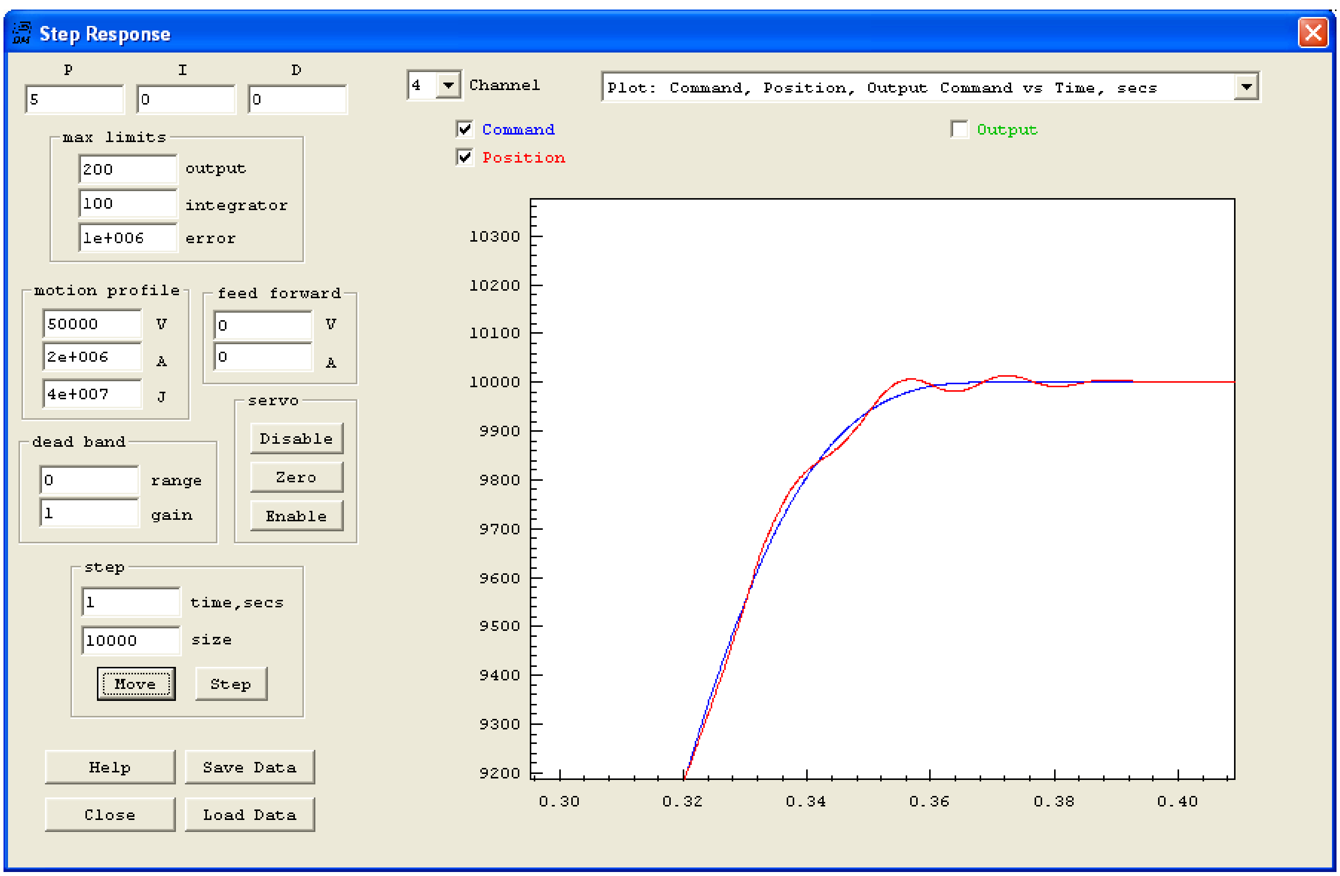
Once the servo loop is functional Servo Tuning should be performed to move the performance, hopefully up to the desired levels. Servo tuning is a complex subject that has been written about extensively. The basic idea is to increase the feedback gains in order to reduce following errors and speed of 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.

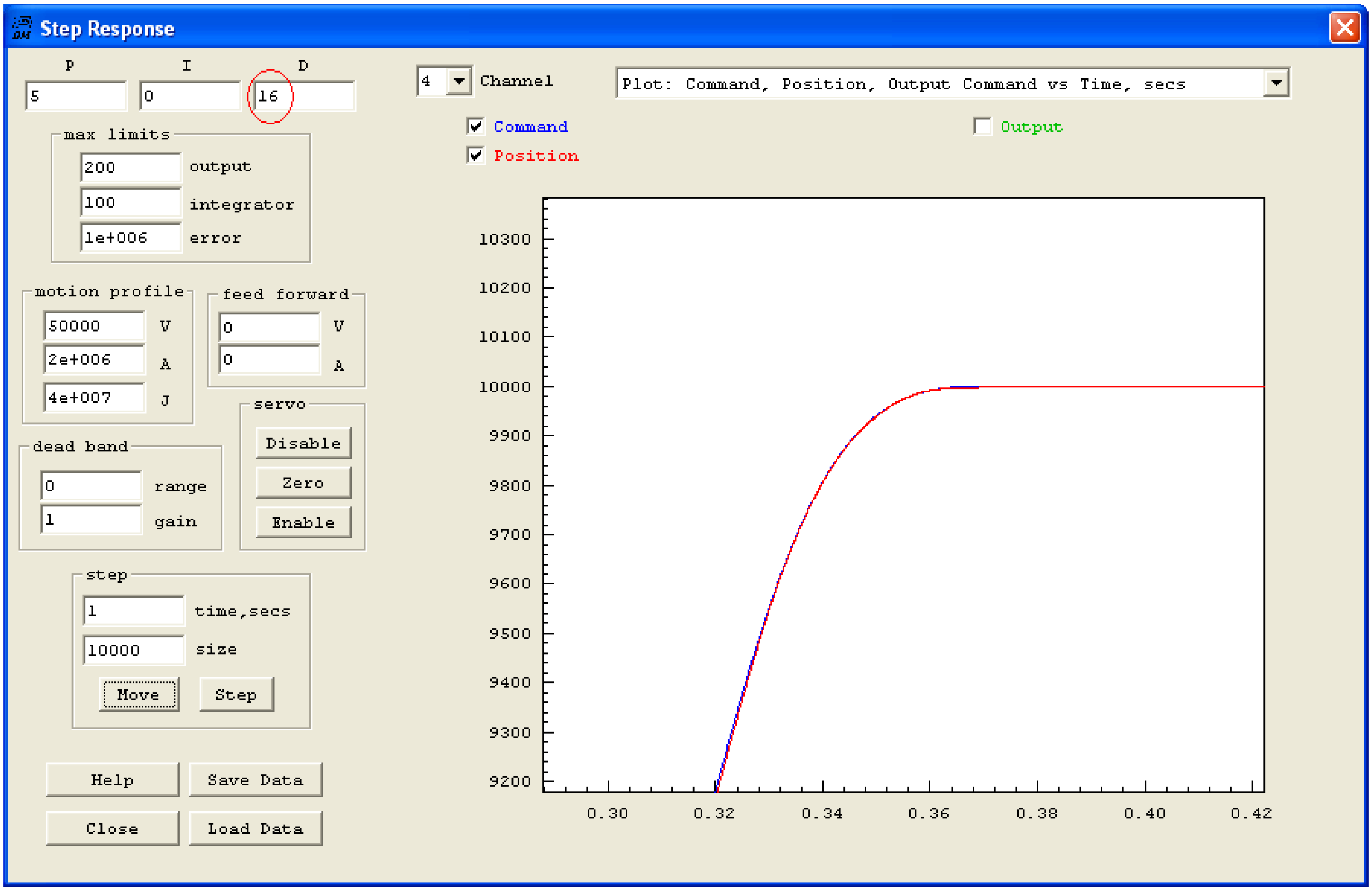


Increasing the gain 500X (from 0.01 to 5) results in a much better response. Notice the blue (desired) plot is almost completely hidden behind the red measured plot. However by zooming in to the purple rectangle (left click mouse drag) we can see oscillation. See the second plot below.

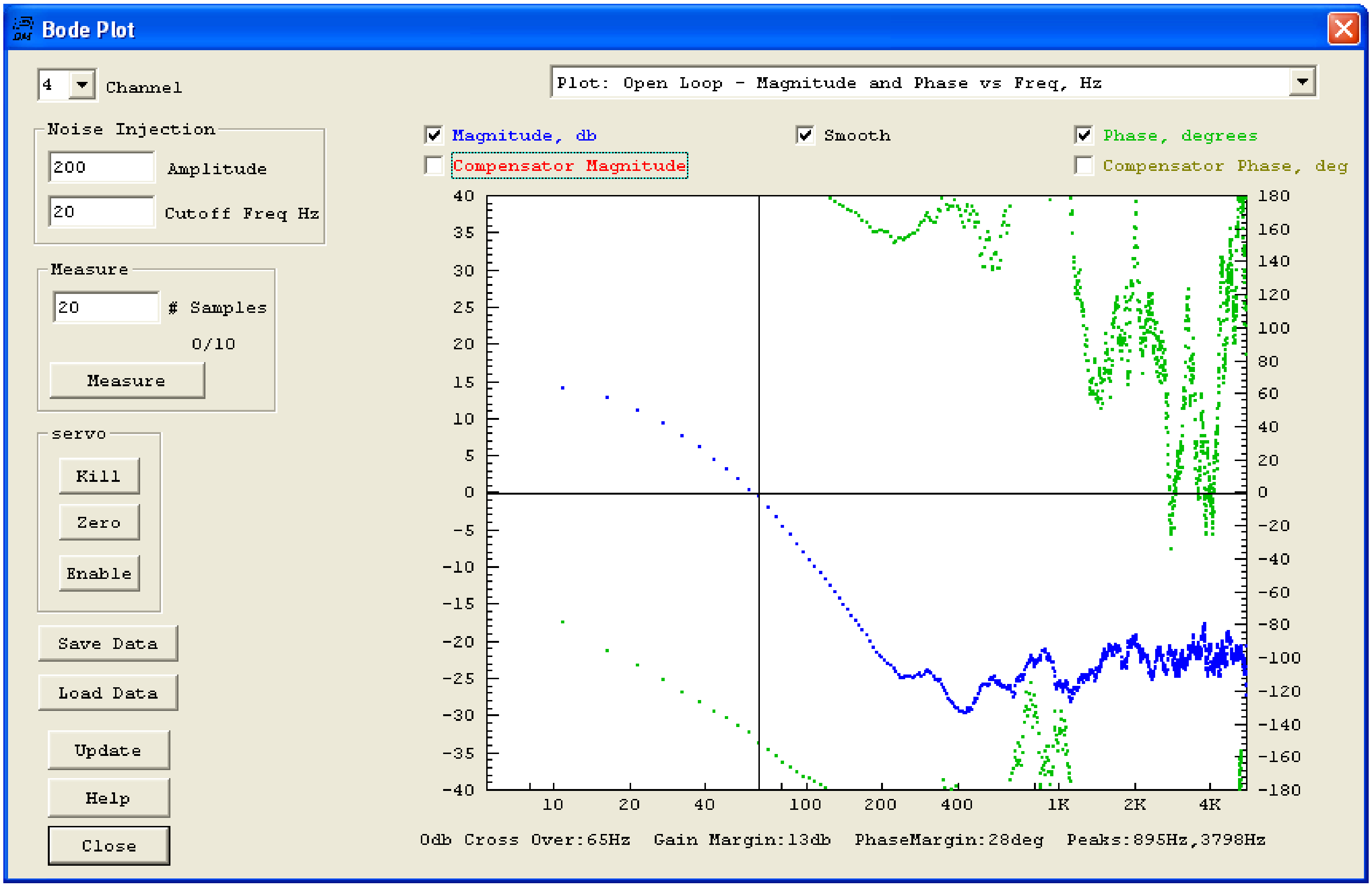

Step Response
P
I
D
5
0
0
max limits
200 output
100 integrator
1e+006 error
motion profile
50000 V
2e+006 A
4e+007 J
feed forward
0 V
0 A
servo
Disable
Zero
Enable
dead band
0 range
1 gain
step
1 time,secs
10000 size
Move
Step
Help
Save Data
Close
Load Data
4
Channel
Plot: Command, Position, Output Command vs Time, secs
Command
Position
Output
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0


Step Response
P
I
D
5
0
0
max limits
200 output
100 integrator
1e+006 error
motion profile
50000 V
2e+006 A
4e+007 J
feed forward
0 V
0 A
servo
Disable
Zero
Enable
dead band
0 range
1 gain
step
1 time,secs
10000 size
Move
Step
Help
Save Data
Close
Load Data
4
Channel
Plot: Command, Position, Output Command vs Time, secs
Command
Position
Output
10300
10200
10100
10000
9900
9800
9700
9600
9500
9400
9300
9200
0.30
0.32
0.34
0.36
0.38
0.40

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.

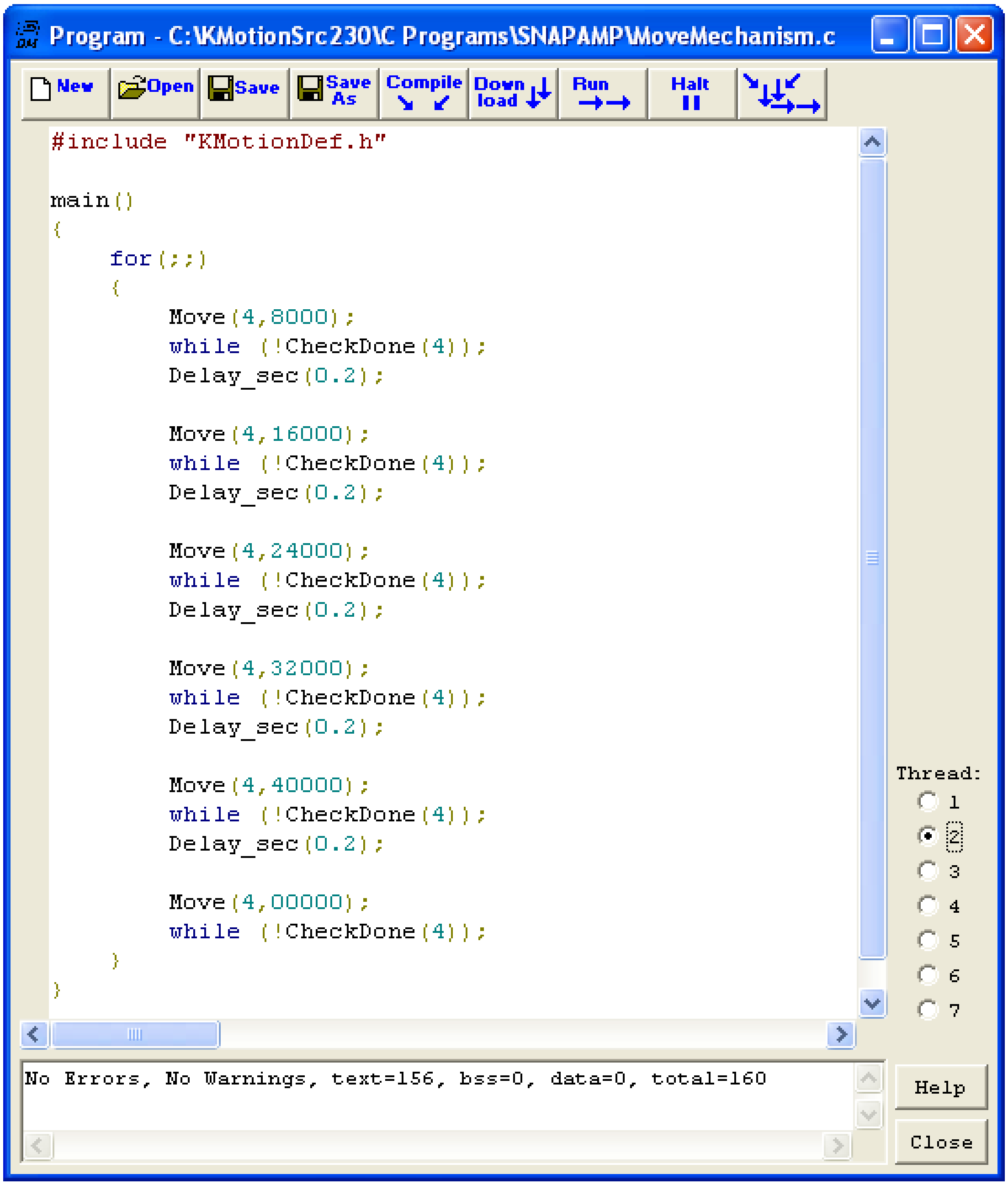


A Bode plot is useful in understanding the frequency response and for designing advanced servo loop filters. Below is the Bode plot response of the system. Important features to look for in a Bode plot is where the Magnitude (blue plot) first passes through the 0 db line. This is the bandwidth of the 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


Bode Plot
4 Channel
Noise Injection
200 Amplitude
20 Cutoff Freq Hz
Measure
20 # Samples
0/10
Measure
servo
Kill
Zero
Enable
Save Data
Load Data
Update
Help
Close
Plot: Open Loop - Magnitude and Phase vs Freq, Hz
Magnitude, db
Smooth
Phase, degrees
Compensator Magnitude
Compensator Phase, deg
0db Cross Over:65Hz Gain Margin:13db PhaseMargin:28deg Peaks:895Hz,3798Hz

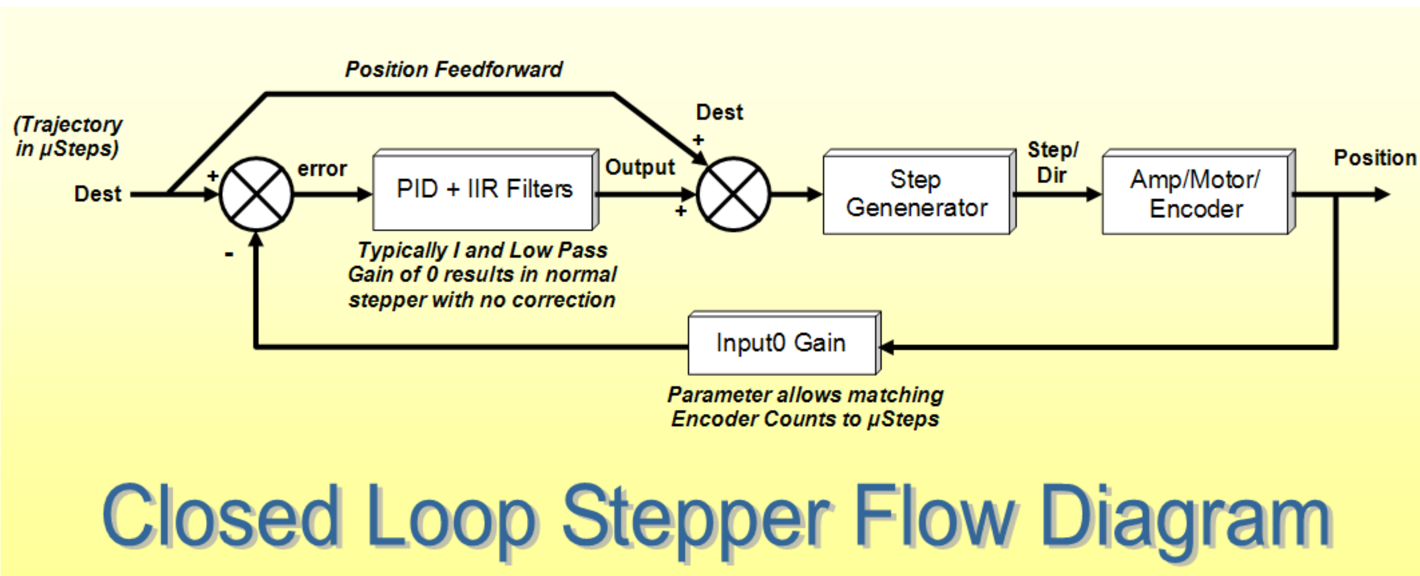
Executing a simple C Program Motion Sequence

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

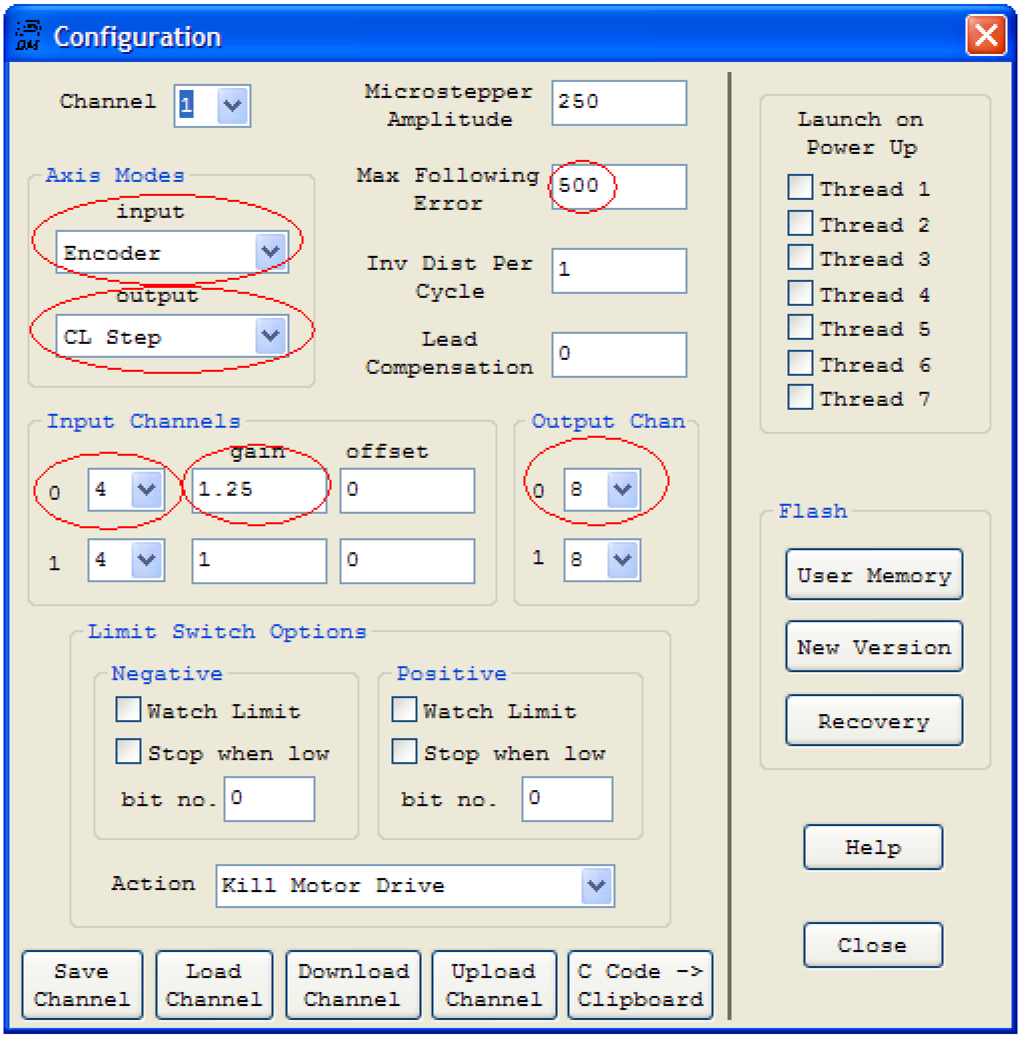
The Closed Loop Stepper Mode works much the same as open loop Step and Direction Output Mode except there is also an error feedback correction term. In fact, if the gains of the correction term (PIDs) are set to zero then the mode will behave the same as an open loop Step Direction Mode. This mode is 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.



To Configure an axis a 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.

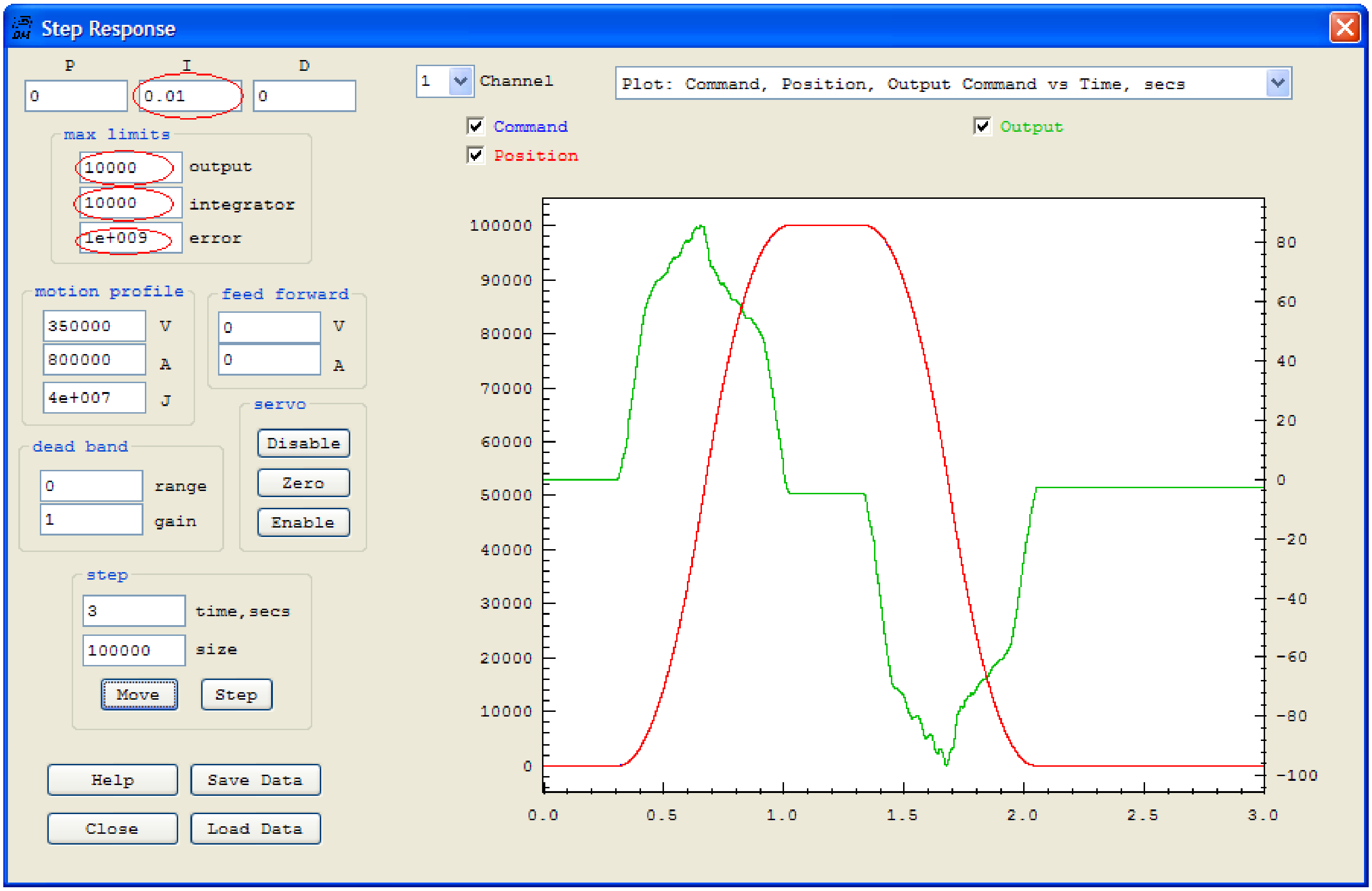
Note that an Input gain of 1.25 is used in this example. This was calculated from the ratio of the number of µSteps/rev to the number of encoder counts/rev. (A micro stepper amplifier set to 50 microsteps/full setp and a standard stepper motor with 200 full steps/rev will have 10,000 µSteps/rev, a rotary 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.

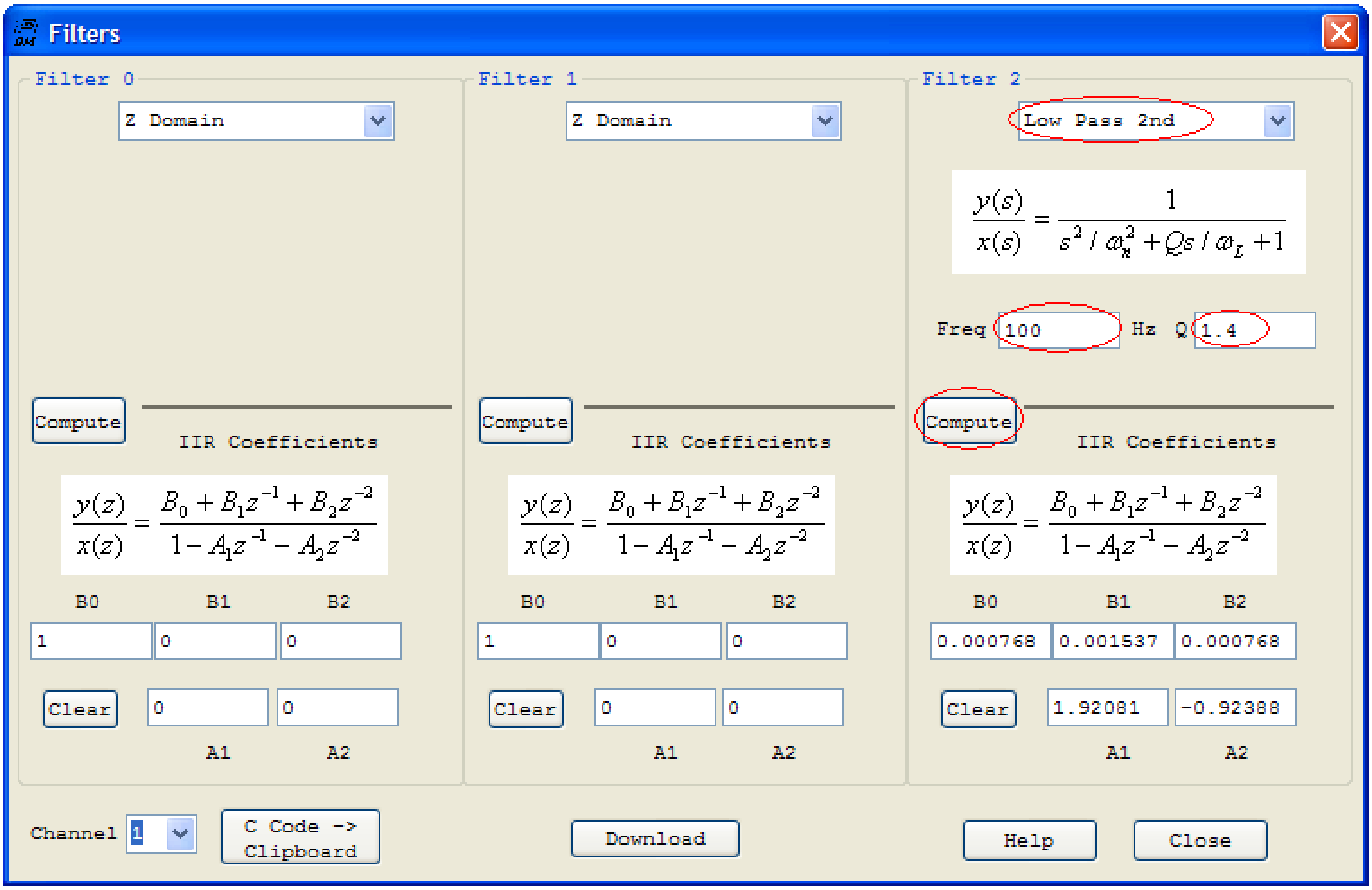


Important PID parameters are shown below circled in red. The I (Integrator) gain of 0.01 is probably the most important. Your system may require more or less. Too much and the system may over shoot or become unstable (oscillate). Too little and corrections will be made slowly. Because we are measure 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.

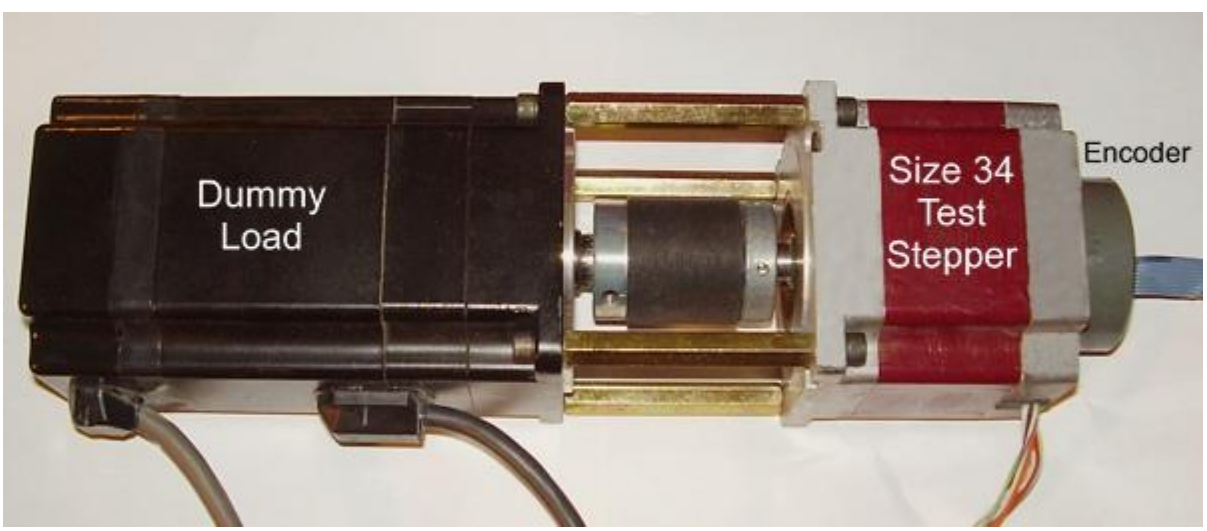
Max limits may also be useful for limiting the correction. In this example the limits are set to large values. Limiting the max error to a small value will limit the maximum slew rate of the Integrator. Max Output and Max Integrator are similar for an Integrator only compensator and will limit the maximum amount of correction that can be made.


Step Response
P
I
D
0
0.01
0
max limits
10000 output
10000 integrator
1e+009 error
motion profile
350000 V
800000 A
4e+007 J
feed forward
0 V
0 A
servo
Disable
Zero
Enable
dead band
0 range
1 gain
step
3 time, secs
100000 size
Move
Step
Help
Save Data
Close
Load Data
1 Channel
Plot: Command, Position, Output Command vs Time, secs
Command
Output
Position
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
80
60
40
20
0
-20
-40
-60
-80
-100
0.0
0.5
1.0
1.5
2.0
2.5
3.0

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 Coeffiecients that are downloaded and used by KFLOP.

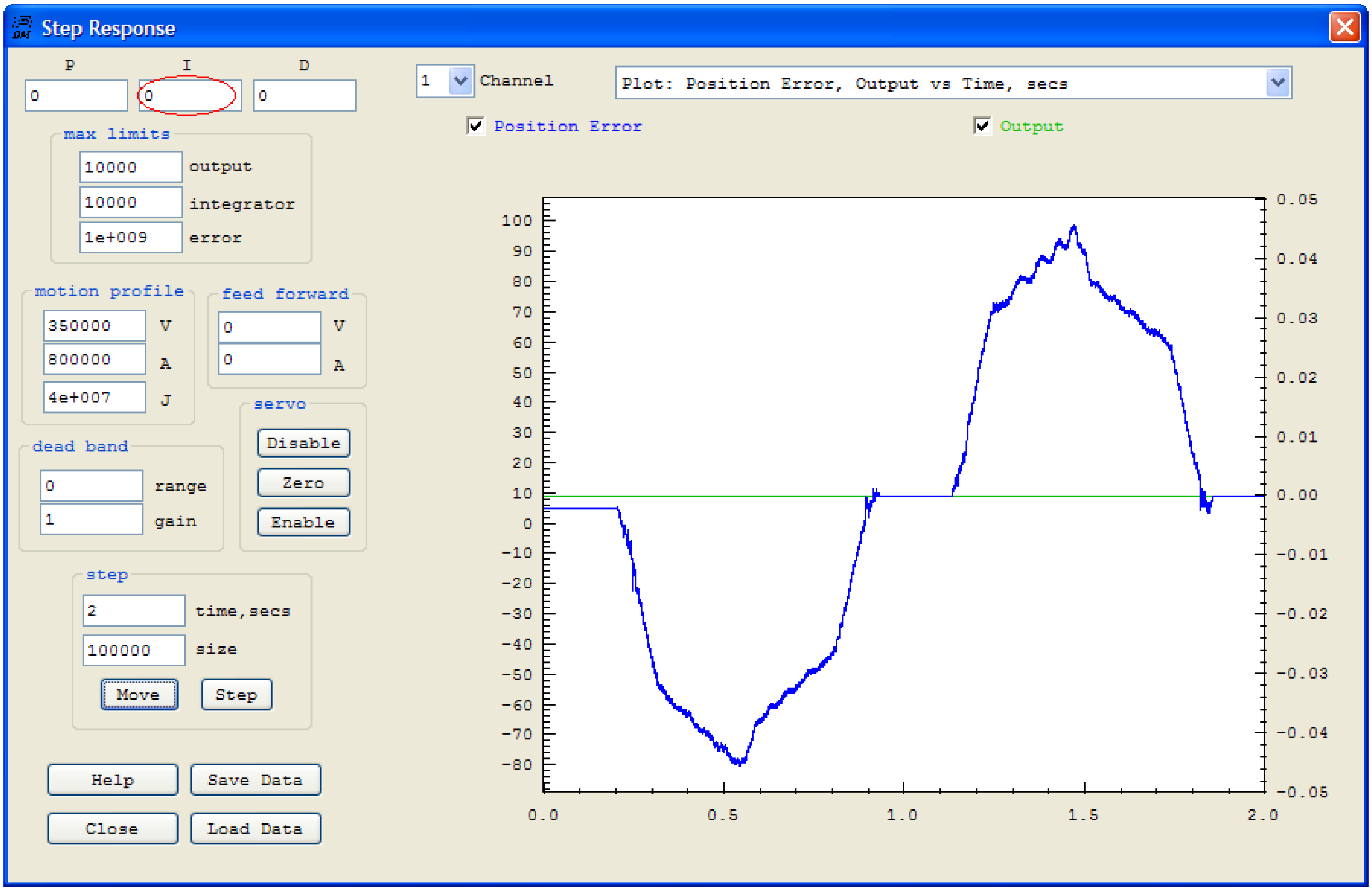


Test Mechanism with Size 34 Stepper with encoder connected to a Dummy Load.



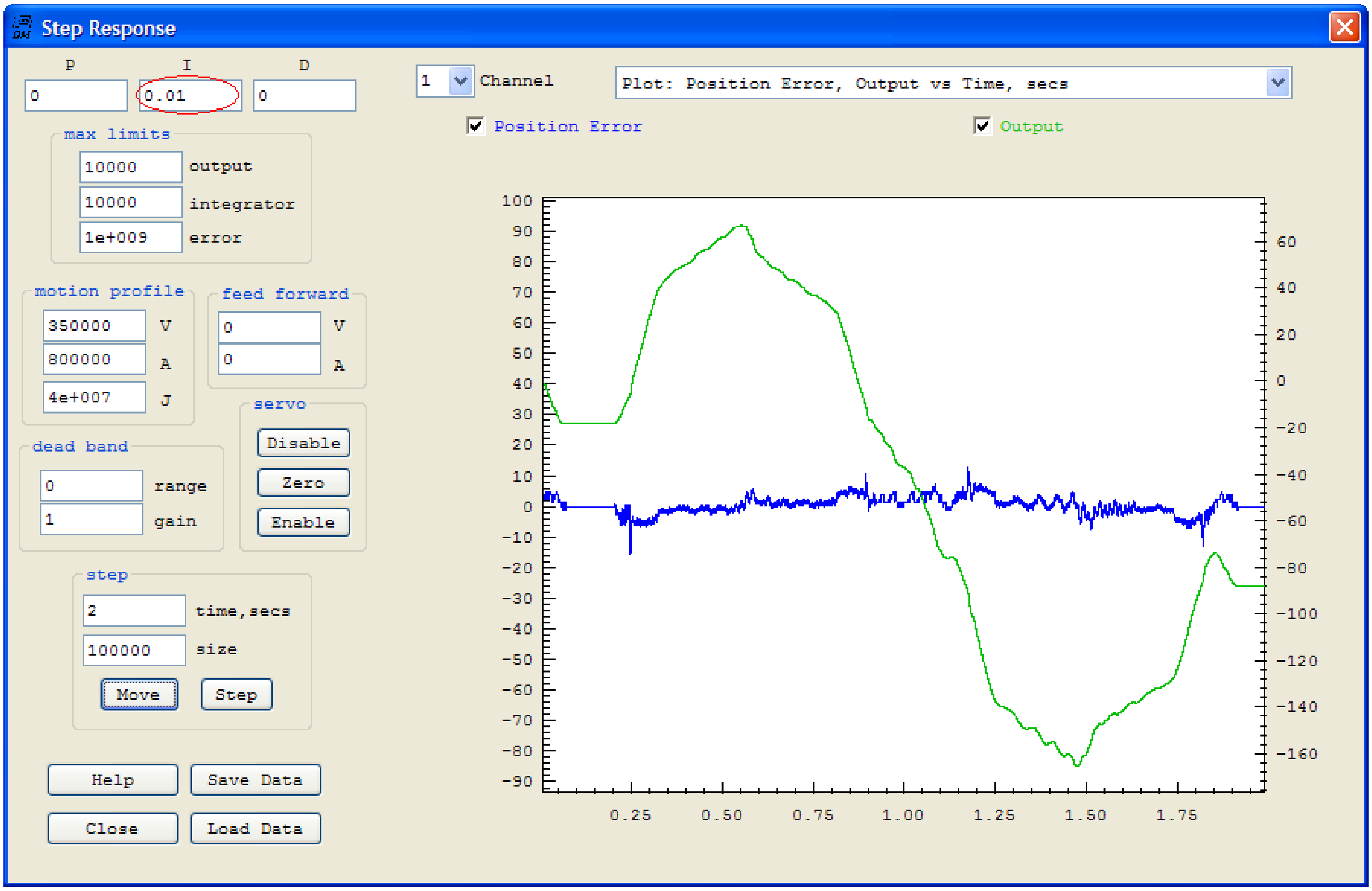
Test Move of a Size34 Stepper with 50 usteps/full step forward and backward at 4000sps

With PID gain zero (no correction). Note encoder shows errors of ~100uSteps



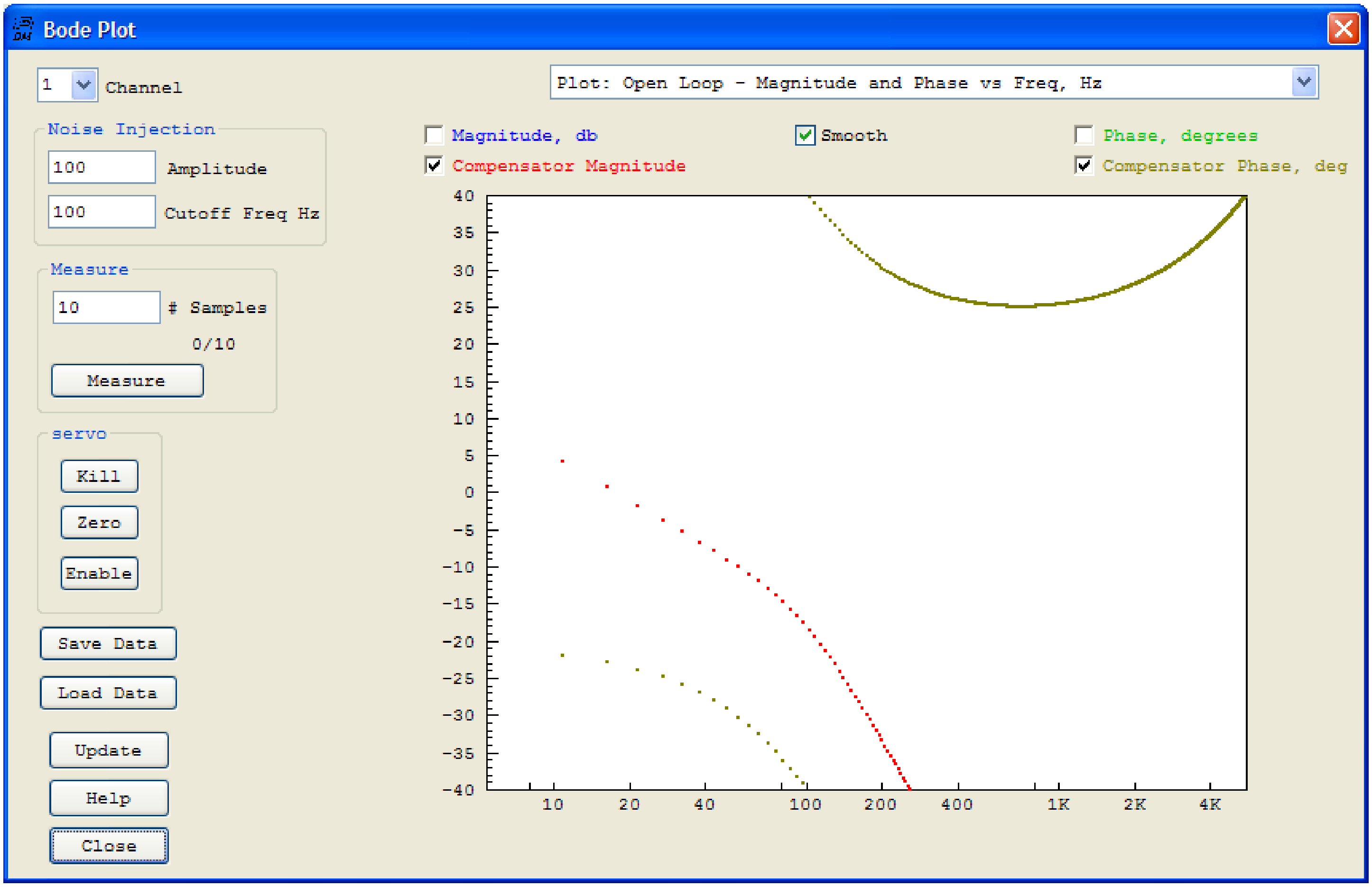
Integrator gain now set at 0.01. Also 2nd Order 100Hz Low Pass Filter Q=1.4 used

Note error is reduced. Blue plot is position error, green is the Output (correction offset)



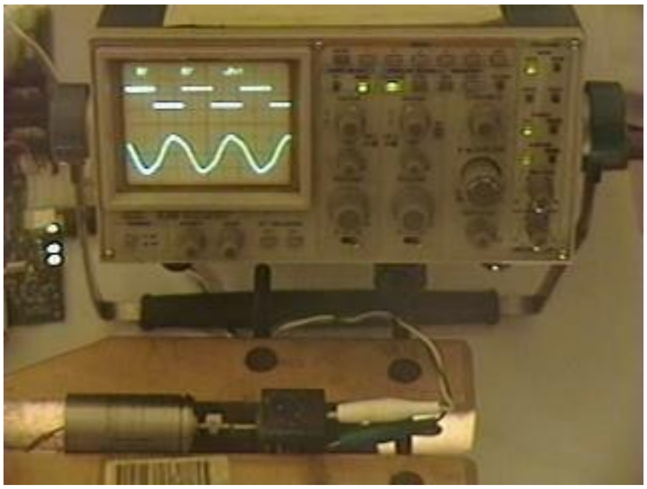
A Bode Plot of the Compensator PID + LP Filter response. I=0.010 and 2nd order Low Pass 100Hz @ Q=1.4. Red plot is Magnitude.

Note that errors less than about 20 Hz will be corrected. Correction gain drops below 1 (0db) at higher frequencies.


Bode Plot
1 Channel
Plot: Open Loop - Magnitude and Phase vs Freq, Hz
Noise Injection
100 Amplitude
100 Cutoff Freq Hz
Measure
10 # Samples
0/10
Measure
servo
Kill
Zero
Enable
Save Data
Load Data
Update
Help
Close
Magnitude, db
Smooth
Phase, degrees
Compensator Magnitude
Compensator Phase, deg
40 35 30 25 20 15 10 5 0 -5 -10 -15 -20 -25 -30 -35 -40
10 20 40 100 200 400 1K 2K 4K

Configuring a Resolver as Input to KMotion

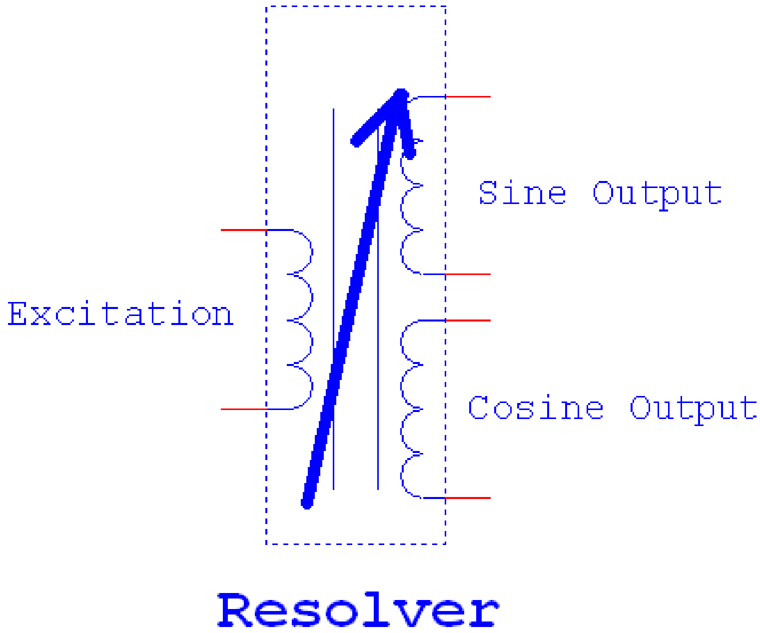
Here is a video overview.



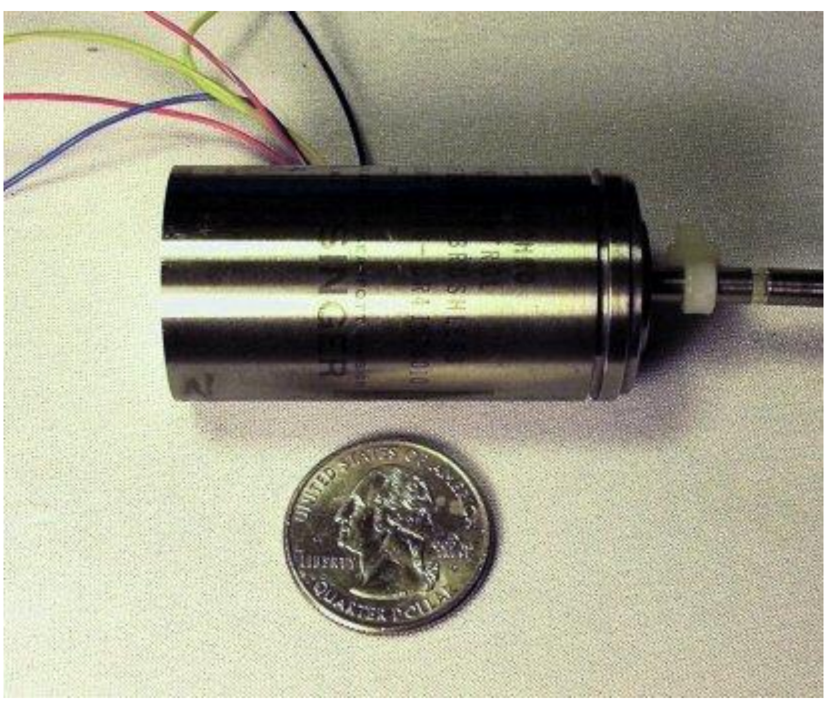
This example configuration shows a Resolver used as a KMotion input device. A Resolver serves a purpose much like a digital encoder. A Resolver is physically somewhat similar to a transformer with two output windings called sine and cosine outputs. For a given excitation input, the amplitude of the sine output is proportional to the sine of the mechanical rotor position, and the amplitude of the cosine output is proportional 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 oils have a resistance of 37 ohms. Swapping the sine and cosine coils will simply reverse the measured motion.



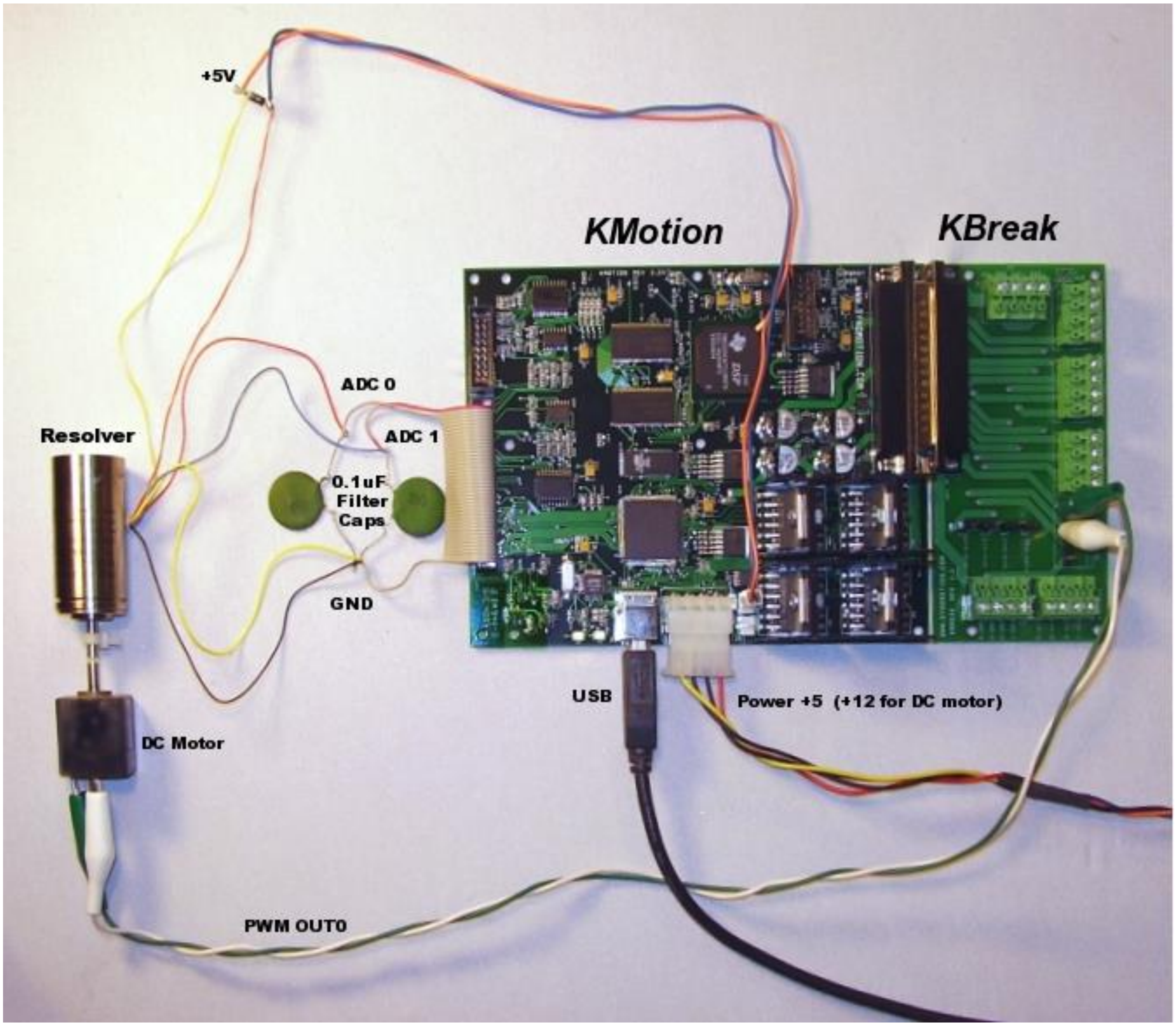
Resolvers may be interfaced to KMotion in one of two ways. One method is where an external circuit or module is available to excite the resolver and determine the sine and cosine magnitudes externally. In this case only the sine and cosine magnitude signals need to be connected to ADC inputs and "Resolver" 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.

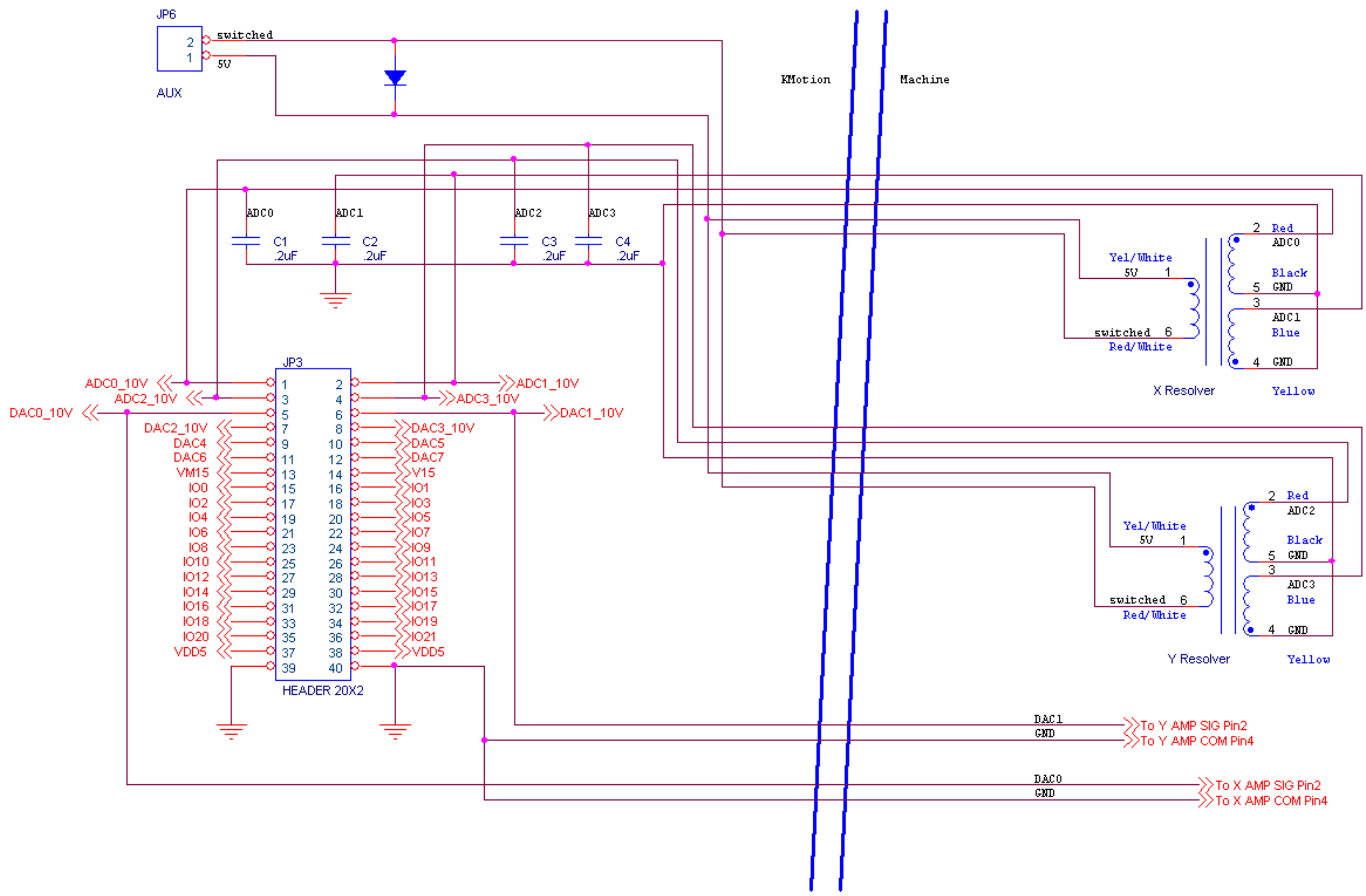
KMotion's Aux Switch output is used in this example to excite the resolver's input coil with a 5V square wave. +5V is applied to one end of the coil and the Aux Switch drives the opposite end of the coil to ground. This applies +5V to the coil which causes current to ramp up in the coil. When the switch 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 processes 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.

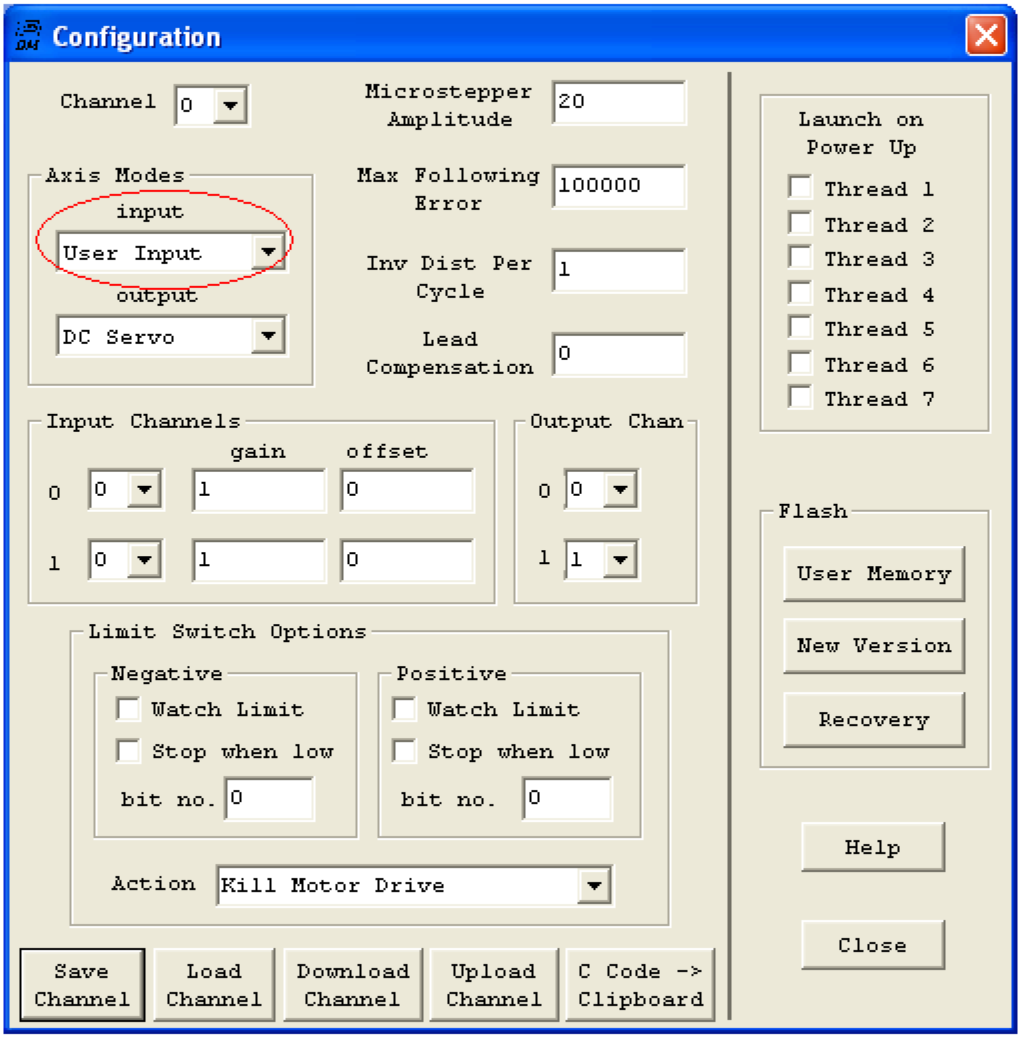


This is a KMotion User Program which basically loops every 4 time slices (720us), switches the 5V excitation, samples the ADC readings, computes the output magnitudes, and calls an internal function (DoResolverInput2) that computes the rotor angle and updates the axis's Position. DoResolverInput2 internally 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.

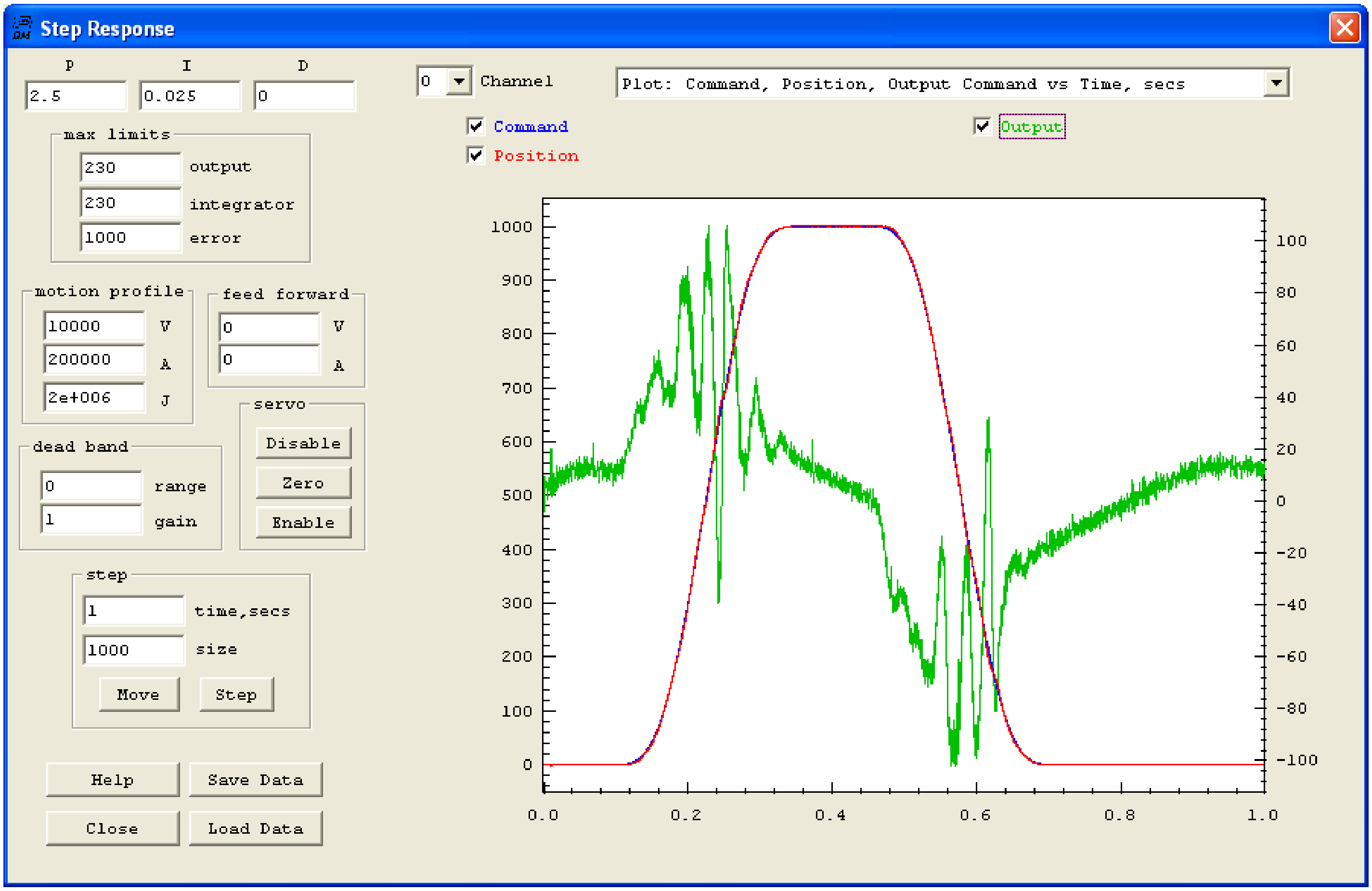
```
#include "KMotionDef.h"

// 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
	}
}
```

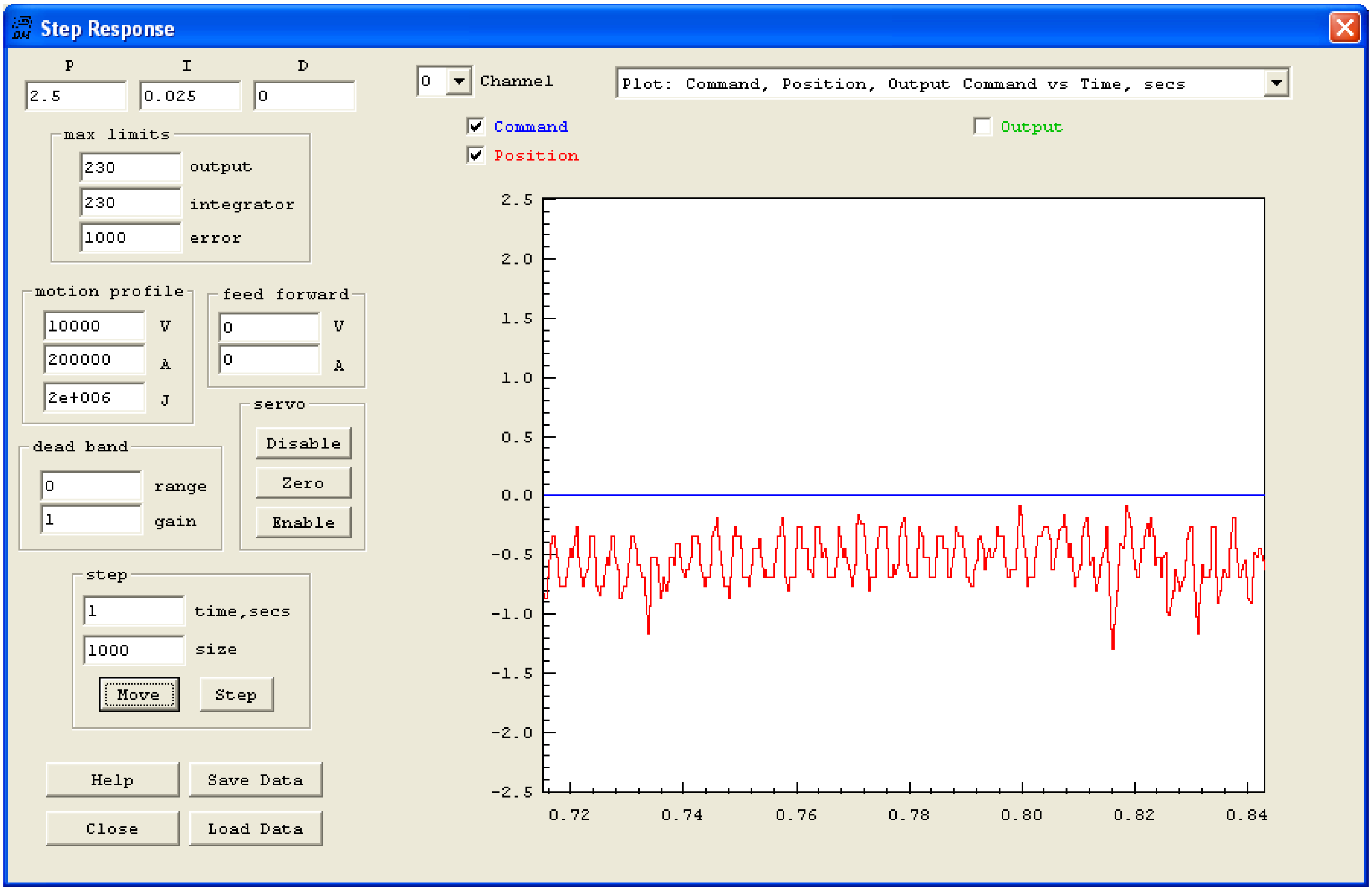
The Configuration Screen input mode is shown here set to "User Input". This allows the User Program shown above to be in charge of setting the current position.



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 (~ 2.5um).


Step Response
P
I
D
2.5
0.025
0
max limits
230 output
230 integrator
1000 error
motion profile
10000 V
200000 A
2e+006 J
feed forward
0 V
0 A
servo
Disable
Zero
Enable
dead band
0 range
1 gain
step
1 time,secs
1000 size
Move
Step
Help
Save Data
Close
Load Data
0 Channel
Plot: Command, Position, Output Command vs Time, secs
Command
Output
Position
2.5
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2.0
-2.5
0.72
0.74
0.76
0.78
0.80
0.82
0.84

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 *)0xfffffffc; // force more than endbuf
gather.endptr = gather_buffer + 3 * n_Samples;

TriggerGather(); // start capturing data

MoveRel(0,20); // Start a motion

while (!CheckDoneGather()) ; // what till all captured

// print all captured data (every 50th sample)
```

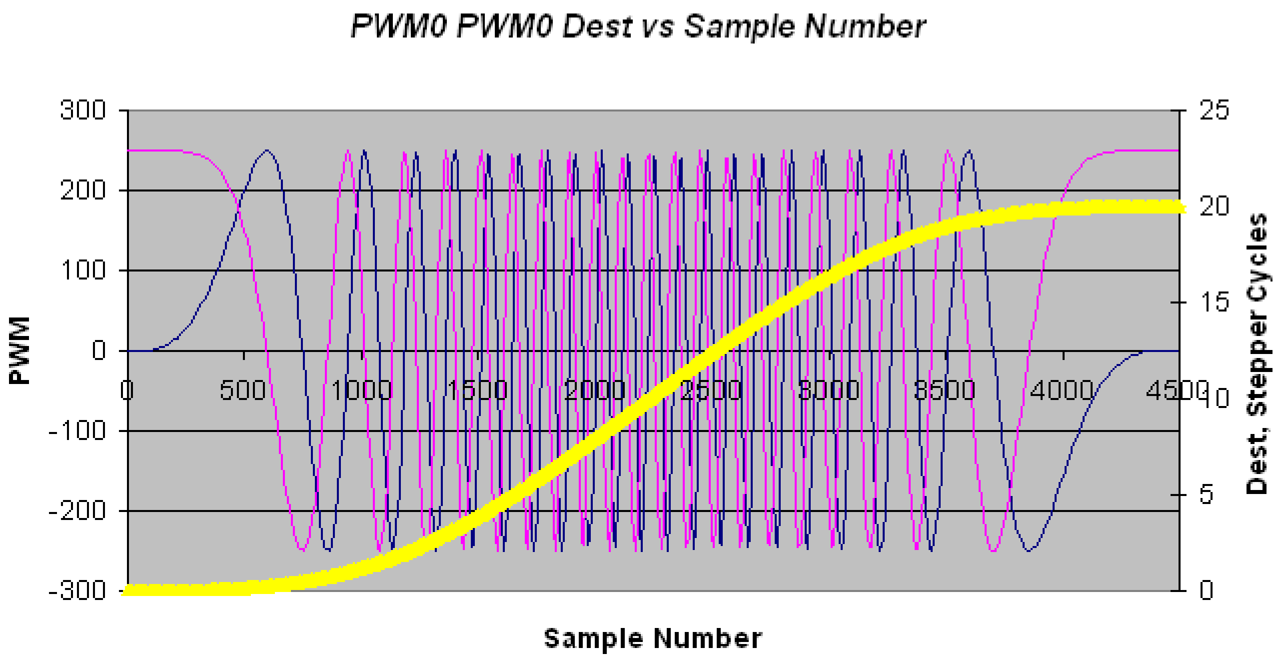
```
for (i=0; i<n_Samples; i+=10)
printf("%d,%f,%f,%f\n", i,gather_buffer[i*3],
gather_buffer[i*3+1],
gather_buffer[i*3+2]);

}
```

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.



Videos

Step and Direction

Brush Motor with SnapAmp

Resolver with KMotion

Nonlinear Kinematics

IR Remote Control

KStep 4-Axis Stepper Amp

Our videos on YouTube

Forum/Support

Dynomotion Forum

CNCzone Forum

Dynomotion Wiki