MN400
Motion Controller
Installation and Programming Manual

Document Revision A8
December 6, 2007

MICROKINETICS CORPORATION
2117-A Barrett Park Drive
Kennesaw, GA 30144
Tel: (770) 422-7845
Fax: (770) 422-7854
www.microkinetics.com

Table of Contents

1 Introduction

1.1 Features

The MN400 is a 4-axis motion controller. It provides pulse and direction signals, general inputs and outputs, and Motion speed scaling. It communicates with a master controller (typically a PC) via RS232 or RS485 serial communications. The MN400 controller can be used with our DM8010, DR8010, DM4050, and UnoDrive or with any third party driver that accepts industry standard step and direction commands. Also available is the MN400E, which comprises of a MN400 controller and an enclosure with wiring, switches, LED's, control potentiometers, and power supply to demonstrate or utilize the controller's capabilities.

The features of the MN400 include:

- RS-232/RS485 and USB interfaces to PC
- Five software selectable baud rates (9600, 19200, 38400, 57600, and 115200)
- Easy to use command set
- High Speed outputs (up to 100Khz step rate)
- Continuous contouring
- Helical Interpolation

1.2 Specifications

Electrical Specifications

Minimum Step Rate	2 pulses/sec
Maximum Step Rate	100,000 pulses/sec
Operating Voltage	5 VDC ±5%
Current Requirements	500mA max.
Outputs	8 Step and 8 Direction optically isolated
Limits	8 Limit inputs, optically isolated
Inputs	8 optically isolated (See Table 3)
Outputs	8 optically isolated
Physical Dimensions	See diagram
Working Temperature Range	32°F ~ 158° F (0° C ~ 70° C)

2 Operation

The MN400 Motion Controller is capable of controlling up to 4 stepper motor drivers that accept industry standard step and direction inputs. The board receives commands via RS232 or RS485 serial connection from a master device, typically a PC or panel mount LCD controller. The commands sent to the MN400 are acknowledged and acted upon. If the MN400 receives an invalid command or it detects an error in transmission, a code will be returned to the master indicating the type of error (see section on return codes below).

2.1 Command Set

The following list contains the commands for the MN400 and a brief description.

All MN400 commands are a single character and are not case sensitive.

Command Format: *Commanddata,data,data,data,data,data*
Data is comma delimited and spaces are not allowed
e.g. m10000,20000,30000,-40000

The numerical values all have a size associated with them. The use of the % character indicates a value not to exceed the range of +/-32767 and the use of the & character provides a range of +/- 2.147 billion. ($2.147X10^9$)

Function	Command	MN400 Command
Arc	A	centerx&, centery&, xdest&, ydest&, dir% Moves tool along the path of an arc (dir: 1 CW, -1 CCW)
BeginCC	(	Begins continuous contouring
ClearExitCode	I	Clears the exit code
Control	C	port%, state% Allows setting or resetting of the output ports
EndCC	)	Ends and executes continuous contouring sequence
ExitCode	/	Retrieve the current exit code
FirmwareRev	?	Returns the current firmware version numbers: "MN400 (Motion processor Firmware Rev#) / (Com Processor Rev#)"

HelicalMove	+	Performs a circular move with a simultaneous linear move on a 3rd axis
Hold	H	mode% (0 – Execute motion commands as soon as received (default), 1 – Hold command for later release via K)
Jogger	J	mode%0 (1 = jog mode, 0 = run mode) Places the controller in jog mode or run mode
JogTapSteps	G	res1&, res2&, res3&, res4&, step_per_inch& Sets the number of steps to dispense upon jog button tap Steps Per Tap = step_per_inch& / res&
JogVelocity	Y	Speed& Sets the speed of jog action when using the jog inputs
LimitStatus	S	Reports the status of the limit inputs
LoadCount	L	m1&, m2&, m3&, m4& Allows modification of the absolute count kept in memory
Mover	M	m1&, m2&, m3&, m4& Performs a vector move
PauseMove	{	Pauses a move in progress
PollDevices	*	Request response from connected device
Profile	F	startspeed&, endspeed&, Accelrate& Sets the ramping profile. Ramping is used during linear moves and jogging.
Program	Z	Enables writing of program to controller memory. Terminate by sending EOF command (^D).
Quit	Q	Abort move that is in progress
ReadCount	N	Returns number of steps not taken during previous move command
ReadPosition	E	Returns the coordinate for each
ReadProgram	R	Read the program stored in controller memory
Release	K	Releases pending motion held with the H command

ResumeMove	}	Resumes the paused move
Retransmit	&	Request retransmission of last message
RunProgram	U	Runs a program stored in controller memory
SetAbsMode	I	mode% (0 - absolute (default), 1 - incremental) Allows you to select absolute or relative mode
SetAddr	=	Sets Controller's ID address until the next power cycle Use in SetConfig program to assign an ID at every power-up
SetBacklash	B	m1lash%, m2lash%, m3lash%, m4lash% Sets the amount of backlash for each axis in steps
SetBacklashDir	D	m1lashdir%, m2lashdir%, m3lashdir%, m4lashdir% Sets the initial state of the backlash direction all parameters are 1 (positive) or -1 (negative)
SetBaudRate	~	rate% Sets the baud rate to the selected value (0 to 4 for 9600, 19200, 38400, 57600, 115200)
SetConfig	.	Enables writing of config program to controller memory. Terminate by sending Quit command (Q).
SetCPlane	P	xmotor%, ymotor%, zmotor% (0 - M1, 1 - M2, 2 - M3, 3 - M4) Selects the X and Y motors to be used for circular interpolation and the Z motor to be used as the traversal axis for helical moves
SetMultiplex	X	multiplex% Sets the resolution for multiplexing linear moves (1, 2, or 4)
SetSpeed	V	startspeed& Sets velocity for linear & arc moves. Enter a negative speed to use external synchronization pulse.
ShieldStatus	[	0 - Open, 1 - Closed
Threader	T	m1&, m2&, m3&, m4&, pulses&, offset% Allows linear moves to be interpolated with external pulses

2.2 Command Set Detail Reference

This section describes the motion controller commands. The format is as follows:

Header	Provides the name of the function, the command character to send, decimal and hexadecimal values of the command character.
Command format	Describes the use of the procedure *and s*hows the proper syntax for calling the procedure.
Returned Values/ Parameters.	Response(s) of the command executed
Example	Shows the use of the routine in a typical code fragment.
See also	Lists related commands that may provide additional details.

ARC A (65) (&h41)

Arc allows the movement of two motors simultaneously in the path of an arc. The Arc routine works with the current selection of any two motors. (See the SetCPlane routine).

Command Format:

A*centerx&,centery&,xdest&,ydest&,dir%*

centerx and *centery* are long integers which represent the center point of the arc. In relative mode, these variables are the distance to the center from the current location. In absolute mode, they are the center point of the circle in absolute coordinates.

xdest and *ydest* are long integers which represent the destination point of the move. In relative mode, these variables are the distance to the destination point from the current location. In absolute mode, they are the destination point of the move in absolute coordinates.

For *xdest* and *ydest* to be valid points, they must specify a point which lies the same distance from the center as the starting point. In other words, the arc must have a constant radius.

dir specifies the direction of rotation. 1 for clockwise, -1 for counterclockwise.

Example:

A-1000,0,-2000,0,1

Will result in a clockwise movement of a half circle of radius 1000.

Notes: The move will be executed at the speed set with the SetSpeed command. If isSetSpeed was called with a -1, the move will be synchronized to the external signal.

If you know the relative angle of an arc move you wish to execute, it is easy to find the X destination and Y destination coordinates that the Arc routine requires. Simply use the following equations:
xdest = R (COS(EA) - COS(BA))
ydest = R (SIN (EA) - SIN (BA))
where R is the radius of the circle, EA is the ending angle, and BA is the beginning angle.

For example, if you wish to move 45 degrees around a circle with a radius of 500, starting from the top of the circle (starting point = 0,500), the following equations would allow you to calculate the X destination and Y destination values required by the Arc routine and execute the move.

```
BA = 90
EA = BA + 45                          ‘ Define the angles.
centerx = 0
centery = -500                        ‘ Distance to the center point.
dir = -1                              ‘ Set direction and delay.
pi = 3.1415926                        ‘ Define pi

xdest = 500  (COS (EA) - COS (BA))    ‘ Calculate Xdest
ydest = 500  (SIN (EA) - SIN (BA))    ‘ Calculate Ydest
```

See also:

SetCplane P

BeginCC ((40) (&h28)

Starts Continuous Contouring mode. This mode is designed to maintain the highest possible speeds for each line segment while observing limitations in spontaneous speed changes, direction changes, and achievable speeds given the length of a segment and required steps to ramp to any speed.

Command Format:

(*margin%*

The margin parameter is optional and sets the number of steps per second to allow the current major axis speed to spontaneously accelerate or decelerate by. The margin defaults to an allowable jump of 20 steps per second in speed, for each axis.

Example:

```
(50
V3000
M500,,500
M505,,500
M509,,500
V3850
M530,,505
M520,,510
)
```

The above example will result in a continuous move at 3,000 steps per second through the first three line segments, ramp up and continue with the next two. During the last segment the controller will ramp down and reach the unramped speed (default 1000) at the end of the last segment.

Notes: When the '(' character is received by the MN400, all subsequent commands are stored in the buffer and are not acted on until a ')', is received. If a 'Q' is received, all previously sent commands are ignored and the mn400 returns to a ready state. The only commands supported within continuous contouring mode are the 'M" move command and the "V" velocity command. The 4th axis is not currently considered in speed calculations. If commanding steps on the 4th axis, during continuous contouring mode, ensure that the 4th axis is not the major axis (axis with the highest number of steps for the move). Once a continuous contouring move is being executed, the Quit and Pause commands can be used to abort or pause the motion. ReadCount and ReadPosition can be used to track position and steps left in the moves.

See also:

EndCC)

ClearExitCode | (124) (&h7C)

Clears the last exit code stored in the mn400.

Command Format:

|

Control C (67) (&h43)

Control toggles the state of the output port(s).

Command Format:

C*port%, state%*

Port is an integer specifying the port to be written to. State is an integer specifying the state to write. (1 or 0) The mn400 has 8 outputs that can be controlled with the 'C' command.

Example:

C1,0
C2,1

Will turn output #1 on and output #2 off.

EndCC) (41) (&h29)

Closes a continuous contouring set of moves. When ‘)’ is received, the mn400 will rapidly calculate speeds for each segment and perform the motion in one continuous move. See BeginCC for more details on the continuous contouring feature.

Command Format:

)

ExitCode / (47) (&h2F)

Returns the most recent exit code from the last move that was performed.

Command Format:

/

Returned Values:

/*Code%*

Where Code is an integer containing the value of the limit switch responsible for halting the last move.

Example:

M5000
/

Will return 0 if the move was successful, or 2 if the move was terminated due to the M1+ limit switch closing.

FeedHold ; (59) (&h3B)

FeedHold controls the reaction to the shield input. When FeedHold is 1, the shield will pause the current move when the shield switch is closed and resume motion when the switch is opened. When FeedHold is 0, closing the shield switch will abort the current move.

Note: The shield switch should be a “Normally Closed” switch so that it is open when the shield is physically closed and closed when the shield is physically opened.

Command Format:

;mode%

Mode is either 1 or 0 representing the desired FeedHold mode.

Example:

;0

Will cause moves to abort when the shield switch is closed.

FirmwareRev ? (63) (&h3F)

Returns the controller model and the revision number of the firmware that is being used.

Command Format:

?

Returned Parameters:

“Controller model (Motion processor Firmware Rev#) / (Com Processor Rev#)”

Example:

?

Returns:

?MN400 V 1.08 / 1.04

HelicalMove + (43) (&h2B)

Performs a circular move on 2 axes while traversing over a third axis.

Command Format:

+centerx&, centery&, xdest&, ydest&, dir%, zdest&, rev%

Where centerx and centery represent the distance from the center of the circle, xdest and ydest represent the stopping point on the final circle move, dir represents the direction of the move, zdest represents the distance to traverse, and rev represents the number of complete revolutions to perform before performing the final circle move. Z traversal direction is controlled by providing a positive or negative number for the z parameter. Dir should be 1 for clockwise or -1 for counterclockwise.

Example:

+-1000,0,-2000,0,1,-500,4

Will perform 4 ½ clockwise rotations, with the x and y axis, while traversing 500 steps in the negative direction of the traversal axis.

Notes: The axes used for helical moves are assigned with the SetCPlane command.

See also:

SetCplane P

Hold H (72) (&h48)

Hold stops the immediate execution of received commands. This is intended to allow a system containing multiple MN400 controllers to receive their commands and when all are ready, a single ‘K’ (Release) command is issued to address 0 to cause all controllers to perform their actions starting at the same time.

Command Format:

H

Example:
The following puts all controllers in hold mode, selects controller #1 and loads it with a vector move, selects controller #2 and loads it with another vector move then slects controller ID #0 (all controllers) and issues a release command.

[Select ID 0]

H

[Select ID 1]

M1000,2000,3000,4000

[Select ID 2]

M9000,2500

[Select ID 0]

K

Will cause controller 1 to execute ‘M1000,2000,3000,4000’ at the same time as controller 2 executes ‘M9000,2500’

See also:

Release K

Jogger J (74) (&h4A)

The Jog command puts the controller in an interactive state with the control panel pushbutton inputs. By activating those inputs (such as with pushbuttons) the motors are turned in the appropriate direction until the button is released.

Command Format:

J*mode%*

Where mode is 1 for jog mode, and 0 for run mode.

Example:

J1

Activates jog mode.

J0

Returns controller to Run mode.

JogTapSteps G (71) (&h47)

JogTapSteps programs the number of steps to dispense with each jog switch tap. There are four possible values that are dispensed. Which is used depends on the two bit hardware selection: Jog0 and Jog1. Jog0 and Jog1 are two electrical connections that are attached to a rotary binary switch or two mechanical switches.

Command Format:

G*jogres1%,jogres2%,,jogres3%,jogres4%,stepsperinch%*

The four jogres values are 16 bit values (short integers) which specity the taps/in. The higher those values are the smaller the movement per tap. The last parameter is the steps per inch actual pitch X logical steps per rev of your machine.

Example:

G2000,1000,100,10,8000

The 1st parameter results in 8000 steps/in / 2000 taps/in = 4 steps/tap (i.e. 0.0005")
The 2nd parameter results in 8000 steps/in / 1000 taps/in = 8 steps/tap (i.e. 0.0010")
The 3rd parameter results in 8000 steps/in / 100 taps/in = 80 steps/tap (i.e. 0.0100")
The 4th parameter results in 8000 steps/in / 10 taps/in = 800 steps/tap (i.e. 0.1000")

The last parameter of 8000 is derived from 400 logical steps/rev * 20 rev/in.

Note: This function retains the fractionals where the numbers don't divide evenly based on the step resolution. These fractions add up and eventually. For example if the step resolution is .00125, setting the divisor to 10 and the steps to 8 would cause each button tap to produce a commanded movement of .0001" even if every 4th command in this case will not translate to actual physical move.

Note that the use of inches in this example can be easily replaced with millimeters or other units of measure that is natural to the leadscrews or translation mechanics of the machine.

LimitStatus S (83) (&h53)

Status returns the current condition of the 8 limit switches and 8 auxiliary inputs.

Command Format:

S

Returns:

S*switches%*

The returned value is that which corresponds to the ON input. In the event that multiple inputs are ON, the returned value is the **sum** of the corresponding values.

Limit Switch or Auxiliary Input	ON Value
Motor #1 – Limit	1
Motor #1 + Limit	2
Motor #2 – Limit	4
Motor #2 + Limit	8
Motor #3 – Limit	16
Motor #3 + Limit	32
Motor #4 – Limit	64
Motor #4 + Limit	128
Aux Input 1	256
Aux Input 2	512
Aux Input 3	1024
Aux Input 4	2048
Aux Input 5	4096
Aux Input 6	8192
Aux Input 7	16384
Aux Input 8	32768

Using the MOD or BIT functions that may be available in your compiler, you may decipher these as necessary.

LoadCounters L (76) (&h4C)

The LoadCount routine allows you to load new values into the position counters in the controller. By loading the variables with zeros, you can clear the count, and thus make the current position absolute zero.

Command Format:

L*m1&,m2&,m3&,m4&*

Where m1 through m4 are 32-bit numbers representing a new step count for that axis.

Example:

L0,-50,0,0

Will set m1, m3 and m4 to 0, and m2 to –50 steps.

Mover M (77) (&h4D)

The Mover routine turns the motor(s) a specific number of logical steps at a given rate. The number of steps can be positive or negative to indicate clockwise or counterclockwise direction. The speed is specified in steps/sec and is set with the SetSpeed command.

Command Format:

M*m1&,m2s&,m3&,m4&*

Where m1 through m4 are long integers representing the destination point of the move for each motor. In relative mode, these variables are the number of steps to the destination point of the move from the current location. In absolute mode, they are the destination point in absolute coordinates. They can be positive or negative to indicate CW or CCW rotation as seen from the back of the motor.

Example:

V1000
M1000,1000,1000,1000
M,,-100

The first move command will step all 4 motors 1000 steps in a clockwise direction at a rate of 1000 steps per second. The second move command will step motor three 100 steps counterclockwise.

See Also:

Readcounters

ARC

Helicalmove

PauseMove { (123) (&h7B)

Allows you to pause a move in progress. The move can be resumed with the ResumeMove command.

Command Format:

{

Example:

```
V1000
M2000
        - wait 1 second -
{
```

Will pause the move at around 1,000 steps and keep track of the remaining steps for a ResumeMove command.

PollDevices * (42) (&h2A)

Causes the controller to return the hexadecimal value &h16 (Ctrl-V) to indicate that there is a controller present. This command can be used to enumerate all of the controllers connected to an RS-485 serial line. Responses from each controller will be delayed based on the Address setting to prevent packet collisions. Since the packet format also includes the board's Address, this command can be used to detect all boards and their Addresses.

Command Format:

*

Example:

*

Will return the following hexadecimal packet:

```
81 2A 03 00 81 2F 81 2A 16 00 81 42
|  |  |  |  |  |  |  |  |  |  |  |---- Checksum
|  |  |  |  |  |  |  |  |  |  |------- Board Address
|  |  |  |  |  |  |  |  |  |---------- Null
|  |  |  |  |  |  |  |  |------------- Present
|  |  |  |  |  |  |  |----------------- Command
|  |  |  |  |  |  |-------------------- Board Address
|  |  |  |  |  |- CheckSum
|  |  |  |  |----- Board Address
|  |  |  |-------- Null
|  |  |----------- Start
|  |-------------- Command
|----------------- Board Address
```

Profile F (70) (&h46)

The Profile command calculates and writes the ramp (acceleration/deceleration) table. This establishes the range of speeds available.

Command Format:

F*startSpeed&,endSpeed&,aclrate&*

Where startSpeed and endSpeed are long integers representing the initial and maximum speeds of ramped moves. Aclrate is a long integer representing the acceleration rate. A typical acceleration rate is in the range of 2000-80000.

Example:

F1000,4000,4000
V2000
M10000

Will result in a staring step rate of 1000 steps/sec and then accelerate to 2000 steps/sec after a period of 1/2 second. The move will then decelerate back to 1000 steps/sec as it completes the 10,000 step move.

Note: The default profile is a start speed of 1000, an end speed of 5000, and an acceleration of 10000 steps/sec/sec. Speeds that would exceed the endSpeed last set with the SetSpeed command will go at the endSpeed. Speeds slower than the startSpeed will go at the speed commanded.

Program Z (90) (&h5A)

Allows you to store a user program in the controller's on-board memory. The controller has 8k of nonvolatile onboard memory, which is shared with both the stored user program and the stored config program. Commands are stored in ascii format. Program mode is completed by sending the Ctrl-D character (shown as hexadecimal '&h04' in the example below).

Command Format:

Z

Example:

Z
V1000
M0,1000
V2000
M0,-1000
(&h04)

Note that the (&h04)is not a string that is sent but a single byte which is the CTRL-D character

Will store a program in onboard memory that will move motor two 1000 steps CW at 1000 steps/sec and then 1000 steps CCW at 2000 steps/sec.

Note: Program can be executed by pressing a switch wired to the "Run Program" terminal on the controller, or by sending a RunProgram command.

Quit Q (81) (&h51)

Aborts a move in progress and halts all motors. If the quit command is sent while buffering move commands in continuous contouring mode, the controller will flush the contouring buffer and return to the normal ready state. The quit command is also used in config mode to signal the end of the config program.

Command Format:

Q

Example:

M1000
Q

Will begin stepping motor one and then abort the move when the Q is received.

ReadCount N (78) (&h4E)

ReadCount returns the number of steps remaining in the current move or a move that has been paused.

Command Format:

N

Returned Parameters:

N*m1&,m2&,m3&,m4&*

ReadCount is called without parameters. After the call to
ReadCount, M1, M2, M3, and M4 will contain the steps remaining for the current move for each of the four motors. These variables are long integers.

ReadPosition E (69) (&h45)

ReadPosition returns the absolute positions of the motors in steps.

Command Format:

E

or

E*measure%*

Where measure is an optional parameter used to enable a speed measurement mode. The measurement mode is used to accurately measure the number of steps per second being output from the controller. If measure is 1, the response will measure steps output within 100ms. If measure is 2, the response will measure steps output within 1000ms (1 sec).

Returned Parameters:

E*m1&,m2&,m3&,m4&*

or

E*m1&,m2&,m3&,m4&,count&*

After the call to ReadPosition, M1, M2, M3, and M4 will contain the absolute locations of the four motors. These variables are long integers. Count represents the number of phantom steps counted during the delay specified with the measure parameter. Count is only returned if the measure parameter was specified. The phantom count represents the number of steps output, on the axis with the largest number of commanded steps, multiplied by the current Multiplex value set with the SetMultiplex command. Since the phantom mode is only used with multiaxis moves, the move being measured must be commanding a move on more than one axis (This works even if the other axis contains only one step). For example, if the measure parameter specified was 1, Multiplex is set to 2 and the count value returned is 300, then the step rate is 1500 steps/sec.

ReadProgram R (82) (&h52)

The ReadProgram command allows you to read the program stored in on-board controller memory.

Command Format:

R

Returned Data:

A line by line listing of the program.

Example:

Z
M1000
M-1000
<&h14> (Ctrl-D)

R

Returns:

M1000
M-1000

Release K (75) (&h4B)

Releases a pending motion that has been put on hold using the Hold command. This is typically sent after the id is set to zero to release all controllers into motion.

Command Format:

K

Example:
The following puts all controllers in hold mode, selects controller #1 and loads it with a vector move, selects controller #2 and loads it with another vector move then slects controller ID #0 (all controllers) and issues a release command.

[Select ID 0]

H

[Select ID 1]

M1000,2000,3000,4000

[Select ID 2]

M9000,2500

[Select ID 0]

K

Will cause controller 1 to execute ‘M1000,2000,3000,4000’ at the same time as controller 2 executes ‘M9000,2500’

Note: If there is no move pending, a START packet will be sent, followed by a NOMOVEPENDING packet.

ResumeMove } (125) (&h7D)

Allows you to resume a paused move. A move can be paused with the PauseMove command.

Command Format:

}

Example:

```
V1000
M2000
{
        - some delay -
}
```

Will pause the move and then continue the remaining steps left in the move when the ResumeMove command is received.

Retransmit & (38) (&h26)

Causes the controller to resend the last message that it transmitted.

Command Format:

&

Example:

?
?MN400 V 1.08 / 1.04
&
&MN400 V 1.08 / 1.04

Note: This command differs from other commands with return data in that it does not send a START packet before sending the message packet.

RunProgram U (85) (&h55)

Run program will simply run the internal program that has been previously downloaded into the controller's memory.

Command Format:

U

Note: The program can be aborted by sending a Quit command, or paused by sending a PauseMove command.

SetAbsMode I (73) (&h49)

SetIncrMode allows you to select absolute mode, or relative mode. In relative mode, all values entered for linear and circular moves are relative to the current location. In absolute mode, the values entered are the absolute locations.

Command Format:

I*modeswitch%*

modeswitch tells the software whether to turn absolute mode on or off. Modeswitch is a 1 for absolute mode, and a 0 for relative mode.

SetAddr = (61) (&h3d)

Sets the controller's internal address. The controller will only respond to messages addressed to the same ID as the internal address or to address 0.

Command Format:

=*addr%*

Where addr is a number between 1 and 127. Address 0 is reserved for communication to all connected controllers.

Example:

```
.
=5
Q
```

The '.' Starts Config mode. '=5' changes the address to 5. 'Q' signals the end of your Config program. This example will make the controller change it's address to 5 each time it is powered on.

Note: The default address is 1. If a SetAddr command is used outside of the Config program, the new address will be reset back to 1 when the power is cycled. The START packet will be sent with the current address. The DONE packet will be sent with the new address.

SetBacklash B (66) (&h42)

SetBacklash sets the amount of backlash for each axis in steps.

Command Format:

B*M1lash%, M2lash%, M3lash%, M4lash%*

The four parameters are short integers representing the amount of backlash (in steps) that is present on each axis.

Example:

B400,0,0,0

Will set 0.05 inches of backlash on the first axis @ 8000 steps per inch.

Note: The default address is 0 for all axes. If a SetBacklash command is used outside of the Config program, the backlash for each axis will be reset back to 0 when the power is cycled. See SetConfig for more details on storing a config program.

SetBacklashDir D (68) (&h44)

SetBacklashDir sets the initial backlash direction for each axis.

Command Format:

D*M1lashdir%, M2lashdir%, M3lashdir%, M4lashdir%*

The four parameters are short integers representing the direction of the last move that took place on each axis. Valid values are 1 and -1.

Example:

D1,-1,1,1

Will cause a positive move on axis 1, 3, and 4, to have no backlash compensation, but a positive move on axis 2 will result in a backlash compensation move prior to the commanded move.

Note: The default backlash direction is 1 for all axes. If a SetBacklashDir command is used outside of the Config program, the direction will be reset back to 1 when the power is cycled. See SetConfig for more details on storing a config program.

SetBaudRate ~ (126) (&h7E)

Sets the controller's serial baud rate. The controller will only respond to messages sent at the controller's baud rate.

Command Format:

~baud%

Where baud is a number between 0 and 4. The following chart shows the baud rate options:

0 – 9600
1 – 19,200
2 – 38,400
3 – 57,600
4 – 115,200

Example:

.
~4
Q

The '.' Starts Config mode. '~4' changes the baud rate to 115,200. 'Q' signals the end of your Config program. This example will make the controller change it's baud to 115,200 each time it is powered on.

Note: The default baud rate is 9600. If a SetBaudRate command is used outside of the Config program, the baud rate will be reset back to 9600 when the power is cycled. A START packet will be sent at the current baud rate. A BAUDCHANGEOK packet will be sent at the new baud rate.

SetConfig . (46) (&h2E)

Allows a user to store a configuration program in the controller's onboard memory. The config program is executed each time the controller I powered on. This command is typically used to assign a new address and baud at every power-up. When the control is powered on, the config program will be executed line for line as if the commands were being sent over the serial line.

Command Format:

.

Example:

.
=5
~4
Q

The '.' Starts Config mode. . '=5' changes the address to 5. '~4' changes the baud rate to 115,200. 'Q' signals the end of your Config program. This example will make the controller change it's address to 5 and baud to 115,200 each time it is powered on.

Note: The config program is stored in the controllers 8k non-volatile memory. The memory is shared between the config program and the user program. The sharing is dynamic, so the larger your user program is, the less room you have for a config program, and vice-versa. All commands can be use in the config. Commands executed from the config script will return the same data that they normally would when sent as live serial commands.

SetCPlane P (80) (&h50)

SetCPlane selects the two motors to be used by the isArc routine.

With an optional 3rd parameter SetCplane can also be used to select the traversal axis for helical interpolation.

Command Format:

P*xaxis%, yaxis%*

Or

P*xaxis%, yaxis%,zaxis%*

Xaxis, Yaxis, and Zaxis are integers ranging from 1 to 4 and correspond to the 4 possible motors M1, M2, M3, and M4. Xaxis may not equal Yaxis.

Note: The default Xaxis, Yaxis, and Zaxis, are 1, 3, and 2 respectively. If a SetCPlane command is used outside of the Config program, the axes will be reset back to 1,3,2 when the power is cycled. See SetConfig for more details on storing a config program.

SetMultiplex X (88) (&h58)

SetMultiplex allows the user to adjust the linear interpolation resolution. With higher resolution, there is less vibration due to velocity ripple along a multi-axis move. There is a tradeoff in that higher multiplexing will require additional CPU overhead and multi-axis moves will have a lower maximum possible speed.

Command Format:

X*resolution%*

Where resolution is 1, 2 or 4. Other values are not valid.

Note: The default multiplex is 1. If a SetMultiplex command is used outside of the Config program, the multiplex will be reset back to 1 when the power is cycled. See SetConfig for more details on storing a config program.

SetSpeed V (86) (&h56)

SetSpeed allows you to set the speed for linear and arc moves, or set the software to use an external synchronization signal.

Command Format:

V*Speed&*

Speed is specified in steps per second. Sending the SetSpeed command with a speed parameter of -1 will put the controller in “external sync” mode. All linear and arc moves will be synchronized to the external signal rather than being executed as a timed move. Since motors 1 and 3 may or may not be stepped for a given step of motor 2, the actual physical speed along the path of the move will exceed the speed parameter.

If you wish to compensate for this, divide the speed by the quantity:

Sqr((M1^2) + (M2^2) + (M3^2) + (M4^2) / MMAX

Where MMAX is the greater of M1, M2 M3 and M4.

For example:

```
M1& = 500: M2& = 1000
M3& = 200: M4& =-200                    'Desired move
MMAX& = M1&                             'Assign to
If M2& > MMAX& Then MMAX& = M2&  'MMAX the largest move
If M3& > MMAX Then MMAX = M3&
If M4& > MMAX Then MMAX = M4&
Speed& = 700                            'desired speed
sconv! = Sqr(M1&^2 + M2&^2 + M3&^2 + M4&^2)/MMAX&
Speed& = Speed& * sconv!                'Speed conversion
```

Note: The default speed is 2000 steps per second.

ShieldStatus [(91) (&h5B)

Reports the current status of the shield input.

Command Format:

[

Returns:

[*status%*

Where status is 0 for open, or 1 for closed.

SwitchStatus : (58) (&h3A)

SwitchStatus returns the status of the Shield, FeedHold, PauseFlag, PauseState, and Abort.

Command Format:

:

Returns:

:ShieldFlag, FeedHold, PauseFlag, PauseState, AbortFlag

The values returned are 0 or 1. The ShieldFlag is normally 1 which represents shield closed. The other flags are normally 0 indicating that it is not in that specific state.

Example:

:

:1,0,0,0,0

Threader T (84) (&h54)

Threader allows you to perform externally synchronized moves in which the external pulses are interpolated with the steps of the move. By attaching an encoder to a rotating shaft and using the A or B outputs for the sync signal, the Threader command can be used to lock up to 4 axes to the incoming signal.

Command Format:

T*M1&, M2&, M3&, M4&, pulses&, offset%*

M1, M2, M3, and M4 are long integers representing the destination point of the move. In relative mode, they are the number of steps to traverse. In absolute mode, they represent the absolute location of the destination point.

Pulses is the number of external pulses that the move will require from start to finish. Since the pulses variable is interpolated with the move, the motors can be made to turn at different speeds, even for a sync signal with a constant frequency. For example, a single axis move of 300 steps with the pulses variable at 1000 will cause the 300 steps to take place during the time it takes for 1000 pulses to occur. If the pulses variable is 2000, the 300 steps will take twice as long to be traversed. This feature is useful for such applications as making screws with different pitches.

Offset is the number of pulses to wait (after receiving the sync signal) before executing the move. This feature is useful for making a multi start screw.

Example:

T500,0,0,0,2000,0

Will cause M1 to move 500 steps. The move will take 2000 pulses to complete.

2.3 Return Codes

The following is a list of codes that are returned by the MN400 upon receipt of a command.

	Return Codes	
	Hex	Decimal
START	&h03	3
MOVE_ABORT	&h04	4
DONE	&h05	5
CHKSUMERROR	&h07	7
INVALIDCOMMAND	&h0c	12
INVALIDPROGCOMM	&h0d	13
PARA_ERROR	&h0e	14
NOMOVEPENDING	&h10	16
MOVEPENDING	&h11	17
BAUDCHANGEOK	&h12	18
TIMEOUT	&h13	19
BUSY	&h14	20
SLAVEBUFFOV	&h15	21
PRESENT	&h16	22
MEMORY_FULL	&h1e	30

2.4 Serial Data Packet Format

The following information is provided so you can write your own communications drivers. However, the MN Library includes routines that handle data transmission and reception without the need to write your own routines.
The data is transmitted in packets from the PC or master controller to the MN400. The packet consists of the address, a string of ASCII characters representing the command and data, and the checksum. The packet format for data being sent to a device is shown below.

Device address | *Command* | *Parameter* | *Device address* | *Checksum*

Note: Spaces are not allowed in the packet (command string).

Device address - address of the device that is to receive the packet. The high bit of the address must be set to 1. The setting of the high bit of the address is handled by the MN Library routines. For the MN400 the device address is always &h81.

Command - a command that is listed in Table 1. Must be one of the commands listed.

Parameter - number required by the command (See Table 1 - Command Summary). The parameter must be in ASCII format with the value between 48 ('0') and 57 ('9'). Also, the data must be within the specified range for the command.

Checksum - a 7-bit checksum that is the sum of all of the characters in the packet up to the checksum. This value is calculated by adding the values for each character in the packet then ANDing the checksum with 127 decimal (7f hexadecimal) to clear the high bit. Calculation of the checksum and clearing the high bit is handled by the MN Library routines.

Example: If sending the command "M1000" to the MN400, the packet in ASCII representation (in hexadecimal) would appear as follows:

```
    M  1   0  0  0
81|4D|31|30|30|30|81|10
  |                 |  |---- Checksum
  |                 |-------- Device Address
  |-------------------------- Device Address
```

The packet format for data being returned from a device is shown below.

Device address | *Command* | *Return value* | *Return code* | *Device address* | *checksum*

Device address - address of the device sending the packet to the PC or master controller. The high bit of the address is set to 1 by the device.

Return value - Numerical value returned only if required by command.

Return code - single character value that indicates if an error has occurred or not (See Table 2 - Return Codes).

Example: If device number 1 is returning 5455,0,0,0 based on the Readcount ("E") command then the packet in ASCII representation (in hexadecimal) would appear as follows:

```
    E  5   4  5  5   ,   0   ,   0   ,   0
81|45|35|34|35|35|2E|00|2E|00|2e|00|06|81|20
  |                                   |  |  |-Checksum
  |                                   |  |-----Device Address
  |                                   |---------Return Code
  |-------------------------------------------------------Device Address
```

The following is an example of a START packet sent after a '*' (Poll Devices) command:

```
81 2A 03 00 81 2F
|  |  |  |  |  |
|  |  |  |  |  |- CheckSum
|  |  |  |  |---- Board Address
|  |  |  |------- Null
|  |  |---------- Start
|  |------------- Command
|---------------- Board Address
```

START packets returned by other command requests would differ only by the Command character and CheckSum. A DONE would look the same except with a 05 in place of the 03 and a different CheckSum.

2.5 Command Responses

Commands that read and report data will first respond with a START packet, and then a packet containing the command character followed by the requested data. With USB communication, START and DONE packets are not sent, so only the command character and requested data will be returned. The table below shows all the commands in this category:

ExitCode	PollDevices	ReadProgram
FirmwareRev	ReadCount	ShieldStatus
LimitStatus	ReadPosition	

Commands that do not return data will first respond with a START packet, and then return a DONE packet. With USB communication there will be no response unless there was an error in the parameters specified in the command. The Table below shows all the commands in this category:

Arc	LoadCount	SetAddr
BeginCC	Mover	SetBacklash
ClearExitCode	PauseMove	SetBacklashDir
Control	Profile	SetBaudRate
EndCC	Program	SetConfig
HelicalMove	Quit	SetCPlane
Hold	Release	SetMultiplex
Jogger	ResumeMove	SetSpeed
JogTapSteps	RunProgram	Threader

JogVelocity SetAbsMode

There are two exceptions to the response rules above:

1. The Retransmit command will not send a START packet. It will only retransmit the last message.
2. If Hold mode is on and there is a move pending, all commands will respond with a START packet, followed by a MOVEPENDING packet.

Note:

The DONE packet is sent for backwards compatibility with MN100 control software. Unlike the MN100, the MN400 sends a DONE packet immediately after the START packet instead of waiting for the completion of a move. This allows queuing of multiple commands in the MN400's onboard RAM without having to wait for a previous command to complete.

2.6 USB Communication

The MN400 will show up as an HID (Human Interface Device) with Vendor ID 0461 and Product ID 0020. The HID driver comes standard with the Windows 98 Second Edition, Windows 2000, and Windows XP operating systems. The HID driver will buffer between 2 and 8 incoming packets depending on the operating system. When the PC's receive buffer is full, the oldest packet in the buffer will be dropped to make room for the most recent packet.

Checksum and addressing are already built into the low level USB protocol, so they are not added to the packet when communicating through USB. In order to maximize speed, START and DONE packets are not sent when communicating through USB. The USB packet size is 64 bytes, but the MN400 will only parse the first 48 bytes of each packet. Multiple commands cannot be sent in a single packet. Each command must be sent as an individual packet.

Sample USB communication code is provided in the USBTerm project. It contains all of the API definitions and routines needed to send and receive data over the USB interface.

3 Installation

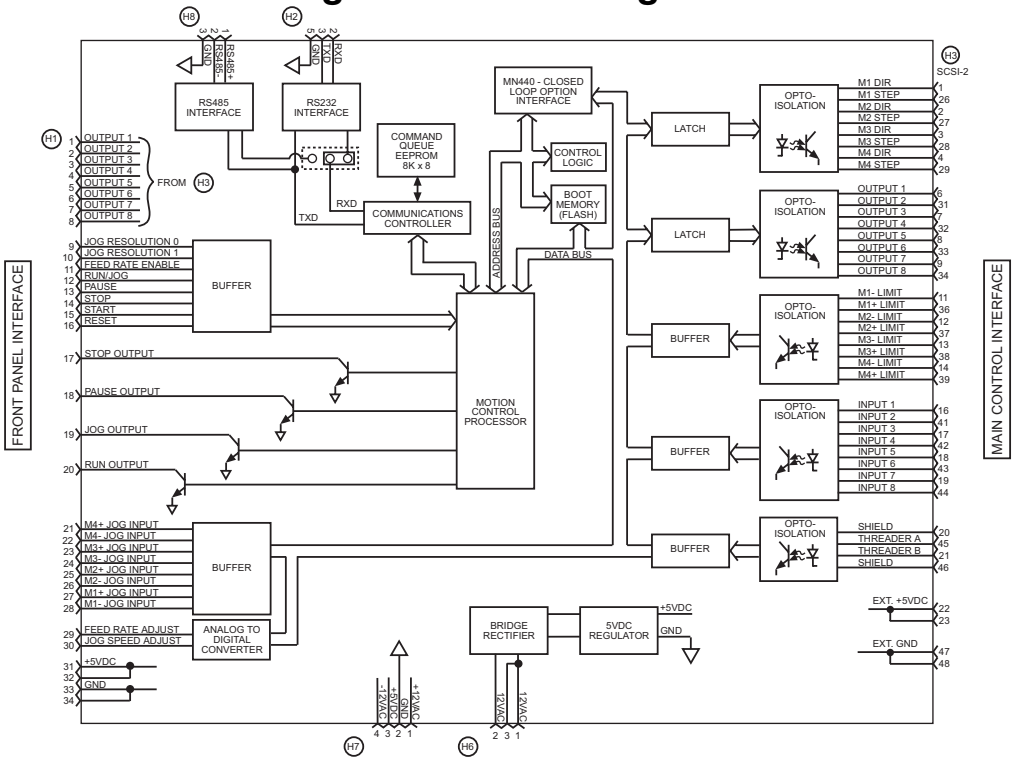
3.1 MN400 Installation

The MN400 has 4 connectors of which 3 must be used for the unit to operate. These connectors include the power, serial, interface and control panel. The power connector is a 4 pin header that connects to a 5VDC supply. The serial connector is a DB9 and connects to a 3-wire serial cable. The interface connector is a 50-pin SCSI-2 connector and it typically connects to the MN401 breakout board. The breakout board contains 2 rows of screw terminals for wiring to the drivers, external high voltage power supply, limit switches, controlled outputs, auxillary inputs and shield. The last connector is a 34 pin IDC header that connects to the control panel. A control panel and using the control panel connector are optional.

3.2 MN400 Block Diagram

The block diagram for the MN400 is shown below. It includes the I/O and pinouts for the board.

Figure 4 - Block Diagram



4 Technical Support

Should you need help in identifying and correcting a problem, the MicroKinetics engineering staff is ready to assist you during business hours. You should refer to the documentation and verify any described adjustments before calling. Be prepared to supply the model number of all components and any software and/or dip switch or jumper settings.

4.1 How to Obtain Technical Support

Technical support is available as follows:

Via Email
Email MicroKinetics with a description of problem symptoms to `mk-tech@microkinetics.com` where it is reviewed and answered daily.

Via Fax
Fax a detailed description of the problem to 770-422-7854 including your fax and voice number. An engineer will call to help you.

Via Telephone
Call our main line directly and request Hardware Tech Support. The number is 770-422-7845.

4.2 Product Service Procedure

If you experience a problem, the technical support staff can determine if the problem requires returning the product for testing and can give you an RMA (Return Merchandise Authorization) number to write on the outside of the package for proper routing. This improves repair turnaround time. They may also provide you with specific packaging information for the protection of the unit during transport.

As a general guideline, when returning an electronic product, always pack in the original antistatic bag. If original packaging is not available, wrap it with aluminum foil and place in a container designed to withstand shipping and handling. Always elect to insure the package with shipping company for the full value.

5 Software

5.1 Windows MNEXAMPLE

An example program, with visual basic 6.0 source, has been included to aid with custom software development. A compiled version has also been included. The compiled version is setup to communicate with an mn100 or mn400 on COM2. The packet handling is done by vbCommLib.BAS which can be included in your own projects. The following describes the 4 main functions of vbCommLib.bas:

mnInitComm - Used to initialize the port and establish communication. An MSComm object must be present on one of your forms, and must be passed to mnInitComm to communicate with.

Format:

mnInitComm(ByRef MsCommObj As Object, ByVal mnport As Long, ByVal mnRate As Long)

Where MsCommObj is the MSComm object on your form, mnport is the serial port to communicate with, and mnRate is the baud rate to communicate at. MnRate is 0-4 representing the baud rate code that the controller uses.

mnXmitPacket - Used to format and send a packet to the controller.

Format:

mnXmitPacket(ByVal Command As String, address As Byte)

Where Command is a string containing a single command, and address contains the address of the controller you wish to communicate with.

mnProcessChar - Used to process incoming response packets coming from the controller.

Format:

mnProcessChar(ByRef mninstr As String, ByRef mnport As Long, ByRef address As Long, ByRef mnerrmsg As String)

Where mninstr is a string that will contain the decoded response from the controller, mnport is the comm port to listen to, address is the address of the controller to listen for, and mnerrmsg is a string where an error message will be placed in the even of an error.

mnCloseComm- Used to close the communication port.

Format:

mnCloseComm(mnport As Long)

5.2 Windows Terminal Program - MNTERMW.EXE

The Windows terminal program (MNTERMW.EXE) provides an interface to the MN400 devices. There are several parts to the terminal program including Command Editor windows, Program Editor windows, and a Device selection window. The Command Editor windows have a Device, Port, Baud, Window, and Help menu when it is active. The Program Editor adds two menus when it is active - Edit and Program. The user can open multiple command editors and program editors and have each window address a different device.

When the MNTERMW program is started it initializes the com port, polls for active devices and opens a command window with the address of the first device found.

NOTE: The MNTERMW program matches its baudrate with the baudrate of the connected devices at start-up. Also, all of the connected devices must be operating at the same baudrate. The power-up baudrate of the MN400 is 9600.

Device Menu

Save	Saves the current configuration for the port and baud rate.
PollDevices	Sends out the poll command (*) to determine which devices are active
Select Device	Allows the user to select a different device address for the active window.
Exit	Exits the program when selected.

Port Menu

The port menu allows the user to select the COM port to use. This value is the first parameter stored in the MNTERMW.DEF configuration file. This file can be changed using a text editor. The values for the ports is shown below.

COM1 - 0
COM2 - 1
COM3 - 2
COM4 - 3

Baud Menu

The baud menu allows the user to select the baud rate to use. This value is the second parameter stored in the MNTERMW.DEF configuration file. The values for the ports is shown below.

9600 - 5
19200 - 6
38400 - 7
57600 - 8
115200 - 9

Edit Menu

Cut	Cuts the selected text from the window
Copy	Copies the selected text to the clipboard.
Paste	Paste the copied text at the current cursor position.
Select All	Highlights all of the text in the active window.
Delete	Deletes the selected text.

Program Menu

These items apply to the active window and the device it addresses.

Read	Reads the program from the device.
Write	Writes the window buffer to the MN400 program memory.
Clear	Clears the window buffer.
Execute	Executes the program stored in program memory.
Home	Returns the motor to the home position.
Stop	Stops program execution (decelerates to stop).

Window Menu

Tile	Tiles the open windows.
Cascade	Cascades the open windows.
Arrange Icons	Arranges icons of minimized windows.
New Command Editor	Opens a new command editor window.
New Program Editor	Opens a new program editor window.
Close	Closes the active window.

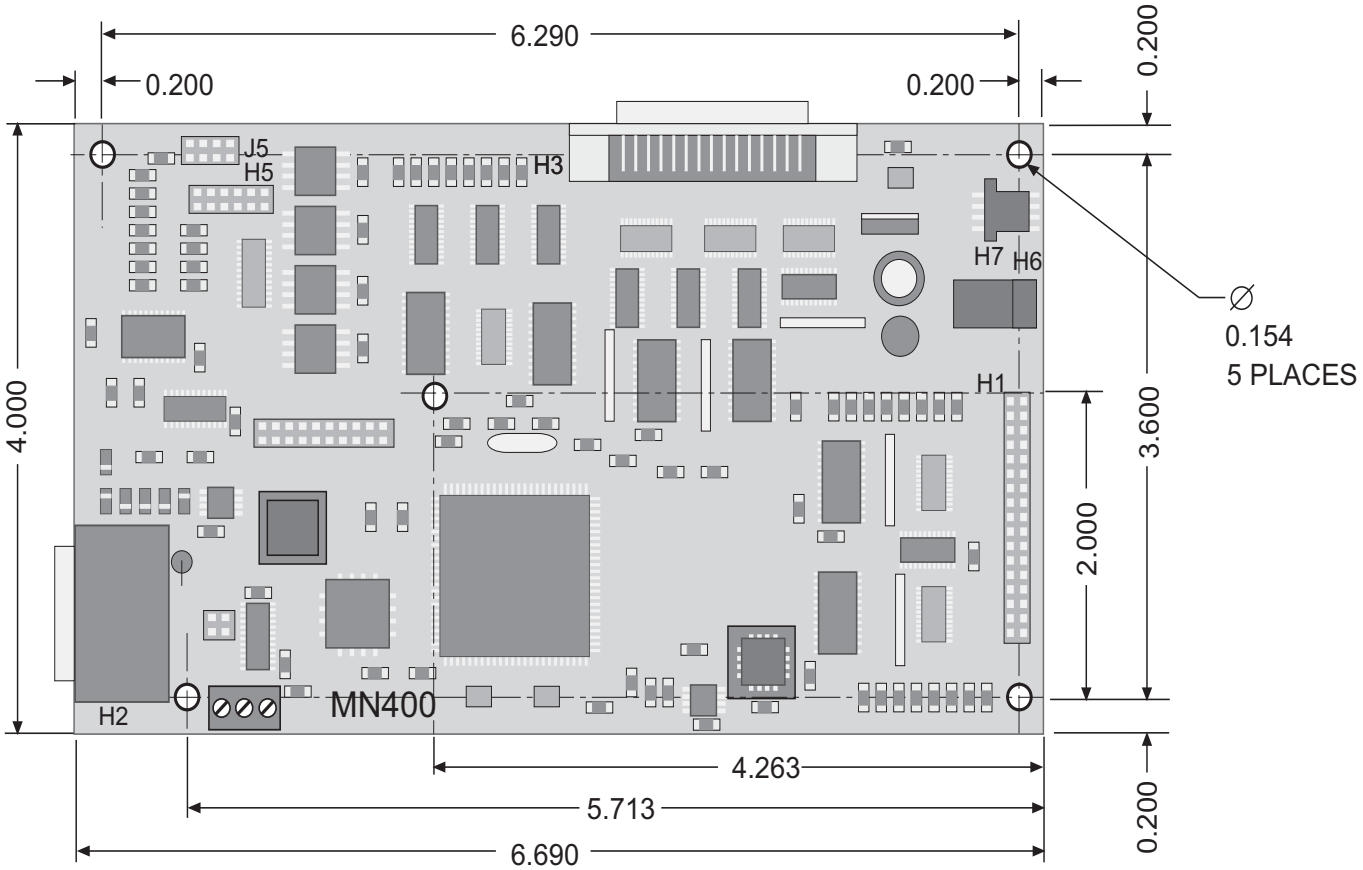
Help Menu

About	Shows about box for program.

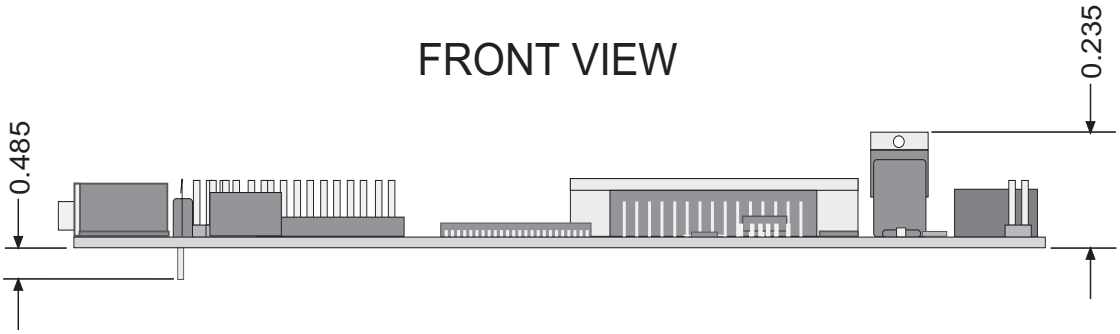
Appendix

Appendix A - Mechanical Drawing

TOP VIEW



FRONT VIEW



Appendix B - Connector Pin Descriptions

H1 - 34 Pin IDC Header Pinout

Pin #	Description	Pin #	Description	Pin #	Description
1	Output 1	13	Pause	25	M2- Jog
2	Output 2	14	Stop	26	M2+ Jog
3	Output 3	15	Start	27	M1- Jog
4	Output 4	16	N/C	28	M1+ Jog
5	Output 5	17	Stop Flag	29	Feedrate Adjust
6	Output 6	18	Pause Flag	30	Jog Speed
7	Output 7	19	Jog Flag	31	+5VDC
8	Output 8	20	Run Flag	32	+5VDC
9	Jog Res 0	21	M4- Jog	33	Ground
10	Jog Res 1	22	M4+ Jog	34	Ground
11	Feedrate EN	23	M3- Jog		
12	Run/Jog	24	M3+ Jog		

H2 - DB9 Serial Port Pinout

Pin #	H2 Description
2	Transmit
3	Receive
5	Ground
7	Ground

H3 - 50 Pin SCSI2 Pinout

Pin #	Description	Pin #	Description	Pin #	Description
1	M1 Direction	18	Aux Input 5	35	N/C
2	M2 Direction	19	Aux Input 7	36	M1+ Limit
3	M3 Direction	20	Shield	37	M2+ Limit
4	M4 Direction	21	Threader B	38	M3+ Limit
5	N/C	22	External +5VDC	39	M4+ Limit
6	Output 1	23	External +5VDC	40	N/C
7	Output 3	24	N/C	41	Aux Input 2
8	Output 5	25	N/C	42	Aux Input 4
9	Output 7	26	M1 Step	43	Aux Input 6
10	N/C	27	M2 Step	44	Aux Input 8
11	M1- Limit	28	M3 Step	45	Threader A
12	M2- Limit	29	M4 Step	46	Threader Index
13	M3- Limit	30	N/C	47	External GND
14	M4- Limit	31	Output 2	48	External GND
15	N/C	32	Output 4	49	N/C
16	Aux Input 1	33	Output 6	50	N/C

17	Aux Input 3	34	Output 8

H6 – RS-485 Port Pinout

Pin #	Description
1	RS485-
2	RS485+
3	Ground

H7 – Power Connector Pinout

Pin #	Description
1	+12VDC
2	Ground
3	+5VDC
4	-12VDC

USB – USB Connector Pinout

Pin #	Description
1	N/C
2	D-
3	D+
4	Ground

NOTE: Do not make any connections to connectors H4, H5, or H8. These are reserved for future expansion.