Peter Norberg Consulting, Inc.
Professional Solutions to Professional Problems
P.O. Box 10987 Ferguson, MO 63135-0987 (314) 521-8808

NCStepper Firmware for the SimStep and BiStep Stepper Motor Controllers

By
Peter Norberg Consulting, Inc.
Matches NCStepper Firmware Revision 1.38-1.51

Table Of Contents

Disclaimer and Revision History

All of our products are constantly undergoing upgrades and enhancements. Therefore, while this manual is accurate to the best of our knowledge as of its date of publication, it cannot be construed as a commitment that future releases will operate identically to this described. Errors may appear in the documentation; we will correct any mistakes as soon as they are discovered, and will post the corrections on the web site in a timely manner. Please refer to the specific manual for the version of the hardware and firmware that you have for the most accurate information for your product.

This manual describes the NCStepper firmware, release version 1.51.

As a short firmware revision history key points, we have:

Version	Date	Description
1.17	1 November, 2002	Initial Beta
1.30	18 November, 2002	24 bit Arc, Idle Status during Arc
1.34	21 November, 2002	Added "A/G aborted" bit to the Latched flags command, and added active skip of '~' received during A/G idle waits, to simplify communications resynchronization issues
1.38	19 December, 2002	First release version.
1.41	15 May, 2003	Changed TTL input levels to CMOS from SCHMITT-TRIGGERED.
1.42	September 12, 2003	Added sense of 1K resistor between SO and GND as method of setting half-power mode at power on.
1.43	October 29, 2003	Allowed setting the microstep size to 1 full step (prior max was ½ step)
	February 21, 2004	Corrected documentation for 'stop oK' command
1.44	December 30, 2004	Corrected documentation about full use the the 'Idle wait' command
	January 3, 2005	Added documentation to better describe timing for K, P and R commands
1.47	February 19, 2005	Added conditional assemblies for many startup defaults, added limit-switch polarity sensitivity (assembly only), added limit switch response sensitivity (instant or 'z')
	April 19, 2005	Synchronized power notes with the UniversalStepper documentation.
1.51	October 31, 2008	Improved serial resynchronization if bad serial data detected

The NCStepper firmware is designed to operate correctly with any of our standard stepper motor controller products. At this point, this includes the SimStepA04, SS0705, BiStepA04, BiStepA05, BiStepA05-1, BiStepA06, and the BiStep2A units.

Product Safety Warnings

All of our board products have components that can get hot enough to burn skin if touched, depending on the voltages and currents used in a given application. Care must always be taken when handling the product to avoid touching these components:

- The 7805 5 volt regulator
- The two SN754410 power drivers (both located near the center of the board for the BiStepA05-1A and BiStepA06 products)
- The two L293D power drivers (both located near the center of the board for the BiStepA04 and BiStepA05 products)
- The two L298 power drivers (located under the large heat sink assembly on the BiStep2A product)
- The PCB board under the SN754410 or L293D power drivers

Always allow adequate time for the board to "cool down" after use, and fully disconnect it from any power supply before handling it.

The board itself must not be placed near any flammable item, as it can generate heat.

Note also that the product is not protected against static electricity. Its components can be damaged simply by touching the board when you have a "static charge" built up on your body. Such damage is not covered under either the satisfaction guarantee or the product warranty. Please be certain to safely "discharge" yourself before handling any of the boards or components.

If you attempt to use the product to drive motors that are higher current or voltage than the rated capacity of the given board, then product failure will result. It is quite possible for motors to spin out of control under some combinations of voltage or current overload. Additionally, many motors can become extremely hot during standard usage – some motors are specified to run at 90 to 100 degrees C as their steady-state temperature.

LIFE SUPPORT POLICY

Due to the components used in the products (such as National Semiconductor Corporation, and others), Peter Norberg Consulting, Inc.'s products are not authorized for use in life support devices or systems, or in devices which can cause any form of personal injury if a failure occurred.

Note that National Semiconductor states "Life support devices or systems are devices which (a) are intended for surgical implant within the body, or (b) support or sustain life, and in whose failure to perform when properly used in accordance with instructions or use provided in the labeling, can be reasonably expected to result in a significant injury to the user". For a more detailed set of such policies, please contact National Semiconductor Corporation.

Introduction and Product Summary

*Please review the separate "**First Use**" manual before operating your stepper controller for the first time. That manual guides you through a series of tests that will allow you to get your product operating in the shortest amount of time.*

The NCStepper firmware is designed to allow our standard dual-stepper motor controllers to operate dual-axis stepper motor systems, wherein the intent is to control the pair of motors as one unit. It supports "perfect line" drawing, wherein the firmware controls the relative X,Y stepping rates needed to cause straight (linear) motion between specified pairs of X,Y addresses. Additionally, the code provides a simplified arc/circle (actually, a polygon) drawing tool, which permits easy drawing of near-circles and similar figures. In order to perform these operations, the firmware makes the assumption that one "step" on the X motor generates the same distance of motion as one "step" on the Y motor. It also assumes that the gear system has no "windup" required when the direction of rotation is changed. That is to say, location 1245 is physically the same when approached from location 2047 as it is from 999 (zero backlash).

The NCStepper firmware shares many of the features of the GenStepper dual-motor controller firmware. The PWM control of the motors is identical, as is the general method of sending numeric parameters for commands. Many of the commands which configure the system are also identical (such as setting the step rate); however, the fundamental control theory is different. The NCStepper firmware explicitly controls both motors at the same time, from a single command (such as Goto or Arc), with automatic step-rate ratioing in order to generate straight lines; while GenStepper explicitly controls the two motors independently, so that one motor may be performing a "slew" operation, while the other is executing a "goto".

The system operates by your first setting up the parameters (such as the next X, next Y, etc.), and then executing a command (such as "G", for "Go to the new X, Y location). As a simple annotated example, the commands given could be as follows (the '*' character is sent by the controller as a "ready" prompt; the rest are commands sent):

```
*0x     - Set X and
*0y     - Y center point for the arc
*1d     - Set the Arc-Delta to 1 degroid (1 degroid is 1/256 of a full circle; or 360/256
degrees)
*256c   - Tell the system that there will be 256 steps to draw (256 small lines)
*0b     - Begin "degroid angle" is 0
*1000a          - Radius of Arc is 1000, and draw. This draws a "circle" of diameter
2000 steps.
*0x     - Reset center back to 0,0 (otherwise, new center would be the last point
drawn)
*0y
*256c   - Reset count to 256; it gets destroyed with each draw
*0b     - Reset arc angle; it is left at last point drawn
*2000a          - Draw a 2000 unit radius circle
*0x     - Once again, go to location 0,0 as center
*0y
*4c     - This time, just set 4 lines in "arc"
*0b     - Again, start at 0 "degroids"
*64d    - set the unit delta to be 64; so that 4 will be a complete "circle"
*3000a          - Draw the 3000 unit radius arc; since done in 4 steps, it is really a
square!
*
```

The above sequence would draw 3 nested figures. The innermost would appear to be a circle, of radius 1000 units. The next would be another circle, of radius 2000 units. The outermost would be a square, rotated 45 degrees, with a diagonal measure of 6000 units.

Short Feature Summary

- Two stepper motors are to be controlled at one time.

- Each motor may be either Unipolar or Bipolar for the BiStep series; they must both be Unipolar for the SimStep (and SS0705) series.
- Limit switches may be used to automatically request motion stop of motion if either motor reaches a limit in either direction.
- Rates of 1 to 62,500 microsteps per second are supported.
- Step rates are changed by linearly ramping the rates. The rate of change is independently programmed for each motor, and can be from 1 to 62,500 microsteps per second per second.
- All motor coordinates and rates are always expressed in programmable microunits of up to 1/64th step. Changing stepping modes between half, full and micro-steps does not change any other value other than which winding pairs may be driven at the same time, and how the PWM internal software is operated.
- Motor coordinates are maintained as 32 bit signed values, and thus have a range of -2,147,483,647 through +2,147,483,647.
- Both GoTo and Arc (actually, multi-line-segment) actions are fully supported.
- Arc vertex locations are calculated to a precision of about 1:10,000,000
- Four modes of stepping the motor are supported:
- Half steps (alternates 1 winding and two windings enabled at a time),
- Full power full steps (2 windings enabled at a time)
- Half power full steps (1 winding enabled at a time)
- Microstep (programmable to as small as 1/64th steps, using a near-constant-torque PWM algorithm)
- A TTL "busy" signal is available, which can be used to see if the motors are still moving. This information is also available from the serial connection.
- Complete control of the motors, including total monitoring of current conditions, is available through the 2400 or 9600 baud serial connection.
- Any number of motors may be run off of one serial line, when used in conjunction with one or more SerRoute controllers.

Firmware Configuration At Time of Ordering Product

As of version 1.47, the NCStepper firmware has a set of initial settings that are selected at power-on or reset *that may be reconfigured at the time the product is ordered*. With the exception of the mode of stepping used when the "Auto-full-step" rate is reached, all of these features may be reset through use of the appropriate serial command. Note that firmware version 2.2 uses the "normal" values shown on this page for these features.

Default Microstep Size

Normally, the firmware defaults to a microstep size of 1/16th of a full step (the equivalent of the "**4!**" command) at power-on or reset. When you order this firmware from us, you have the option of setting this to any of the valid values (1/64, 1/32, 1/16, 1/8, ¼, ½ or full-step).

Default Stop Rate

Normally, the firmware defaults to a stop rate of 80 microsteps per second at power-on or reset (equivalent to the "**80k**" serial command). This can be ordered as any valid stop rate for the system.

Default Ramp Rate

Normally, the firmware defaults to a ramp rate of 8000 microsteps/second/second (equivalent to the "**8000p**" command). This can be ordered as any valid ramp rate for the system.

Default Full-Power Level (No 1K resistor on SO)

Normally, we ship the product such that the default code will select full winding current operation (see the "**0H**" command) when the board is reset or powered on and there is no 1K resistor installed between the SO signal and GND. At the time of ordering the product, you may change this to operate in ½ power mode ("1H") in this case.

Default Low-Power Level (1K resistor installed on SO)

As with the Full-Power-Level, we also provide an automatic selection of ½ power level (approximately) at the time of board reset (equivalent to the "**1H**" command). This mode may be configured by inserting a 1K resistor (1/4 or 1/8 watt) between the SO TTL output signal and GND. You may optionally order this to be the full power level ("0H") if this is better for your application.

Note that for both the high and low power level defaults, the actual current level used can be redefined at any time through use of the "h" command.

Default Motor Idle Winding Current

Normally, at power on or reset, the motor windings are set to be off (no current supplied) whenever motion has completed (equivalent to the "**0W**" command). At the time of ordering the product from us, you may specify the default idle winding mode to be any of our valid values (see the "W" command for details).

Default Limit-Switch Stop Mode

Normally, the firmware defaults to treating a limit-switch input as 'soft'; that is to say, the firmware issues a 'z' command when a limit is reached. This can be ordered as a 'hard' stop – the board will **INSTANTLY** stop the motor when a limit is reached. Note that damage to gear trains is possible if this option is ordered!

Hardware Configuration

The NCStepper firmware has two features that can be configured as startup options. This means that any combination of these features may be automatically controlled whenever the firmware receives a power-on, hardware reset, or software reset action. Both features are selected by adding an external 1K resistor to ground a specific TTL output pin.

Configuring Serial Baud Rate

By default, all serial communications with the NCStepper firmware operate at 9600 baud, 8 data bits, 1 stop bit, no parity. If you need to communicate at 2400 baud, you may connect a 1K resistor (1/4 or 1/8 watt) between the RDY TTL output signal and GND.

Please refer to the following section entitled "Labeling Of Board Signals" for information on where to find these signals.

This feature is available in all versions of the NCStepper firmware.

Configuring Half-Power Mode (equivalent to the "H" command)

Half-Power mode allows you to operate motors at higher voltages, while still operating at their nominal current. This can allow you to either operate motors whose nominal voltage is otherwise too low for our products, or to force motors to be able to operate at higher speeds. **Determining the correct voltage to use is a non-trivial task; please see the separate manual "Half Power Notes" for full details about this option before attempting to use it!**

This mode may be configured by inserting a 1K resistor (1/4 or 1/8 watt) between the SO TTL output signal and GND. The hardware selection may be changed at any time through issuing the "1h" or "0h" commands, as described elsewhere in this manual. However, by operating through use of this hardware strap, you are much less likely to ever "blow out" a board by failing to issue the "1h" command after a power-on or reset condition!

Please refer to the following manual section entitled "Labeling Of Board Signals" for information on where to find the SO signal.

This hardware strap is available on firmware versions 1.42 and later.

Power-On (and reset) Defaults

In addition to the above hardware straps, the board acts at power on (or reset) as if the following serial commands have been given:

- **2=** – Select the next "G"oto assigns the location instead of moving the motors
- **0X** – Set next X will be 0
- **0Y** – Set next Y will be 0
- **G** – Reset both motors to be at location 0
- **0H** – Set motors to full power mode
- **80K** – Set the "Stop OK" rate to 80 microsteps/second
- **3O** – Set the motor windings Order to "microstep"
- **8000P** – Set the rate of changing the motor speed to 8000 microsteps/second/second
- **800R** – Set the target run rate for the faster motor to 800 microsteps/second
- **1V** – Set <CR><LF> sent at start of new command, no transmission delay time
- **0W** – Full power to motor windings

Labeling Of Board Signals

There are currently a total of 6 significant versions of the SimStep and BiStep series of boards available. These versions can be roughly grouped into two major releases: Release 1, which has a large (19 pin) SIP header in the middle of the board which contains all of the standard TTL I/O signals, and Release 2, which uses clearly labeled connectors at the edge of the board which contain the same signals.

Release 1 Pinout for J1– SimStepA04, BiStepA04, and BiStepA05

The pinout for the J1 connector on the release 1 set of boards (the SimStepA04, BiStepA04, and BiStepA05) is as follows, counting from the "top" part of the connector (nearest the DB9 serial connector) on down. Note that this connector is the 19 pin SIP header mounted in the middle of the board.

Pin	Board Label	Signal as used in this manual
1	RTC	
2	+5	
3	RST	
4	GND	GND
5	GND	GND
6	A0	LY-
7	A1	LY+
8	A2	LX-
9	A3	LX+
10	B0	Y-
11	B1	Y+
12	B2	X-
13	B3	X+
14	B4	NXT
15	B5/READY	RDY
16	B6/SERIN	SI
17	B7/SEROUT	SO
18	+5	
19	GND	GND

Release 2 – BiStepA06, BiStep2A, and SS0705

The Release 2 serials of boards are fully labeled. The signal names can be found by looking on the board near each connector. Note that there are 3 input connectors, which contain equivalent signals to those from the J1 connector on the earlier release.

- LIM, which contains RST, LY-, LY+, LX-, and LX+
- SLEW, which contains Y-, Y+, X-, and X+
- IO, which contains NXT, RDY, SI, and SO

Input Limit Sensors, lines LY- to LX+

Lines LY- through LX+ are used by the software to request that the motors stop moving when they reach a hardware-defined positional limit.

The connections are:

Signal	***Limit Sensed***
LY-/A0	-Y
LY+/A1	+Y

LX-/A2	-X
LX+/A3	+X

The connections may be implemented as momentary switch closures to ground; on the connector, a ground pin is available near the LY- pin. They are fully TTL compatible; therefore driving them from some detection circuit (such as an LED sensor) will work. The lines are "pulled up" to +5V with a very weak (10-20K) resistor, internal to the SX-28 microcontroller.

The voltages which are detected as a logic "0" or "1" depend upon the firmware version:

- On firmware versions 1.38 and earlier, voltages which are <=0.8 volts are considered to be "0", while voltages >=4.2 volts are "1". Voltages in the range of 0.9 to 4.1 are transitional, and are ignored by the processor.
- On firmware versions 1.39 and later, voltages which are <=2 volts are considered to be "0", while voltages >=3 volts are "1". Voltages in the range of 2.1 through 2.9 are transitional, and will be randomly treated as "0" or "1" by the processor. Note that any connection which works with version 1.38 will still work with version 1.39 and above.

The stop requested by a limit switch normally is "soft"; that is to say, the motor will start ramping down to a stop once the limit is reached – it will **not** stop instantly at the limit point (unless a special firmware option is ordered). Note that if a very slow ramp rate is selected (such as changing the speed at only 1 microstep per second per second), it can take a very large number of steps to stop in extreme circumstances. It is quite important to know the distance (in microsteps) between limit switch actuation and the hard mechanical limit of each motorized axis, and to select the rate of stepping ("R"), rate of changing rates (the slope, "P"), and the stop rate ("K") appropriately.

As the most extreme example possible:

- if for some insane reason the motor is currently running at its maximum rate of 62,500 microsteps per second,
- and the allowed rate of change of speed is 1 microstep per second per second,
- and the stop rate was set to 1 microstep per second,
- then the total time to stop would be 62,500 seconds (a little over 17.3 hours -- groan!), with a distance of ½ v^2, or ½ (62,500)^2, or 1,953,125,000 microsteps.
- Note that this same amount of time would have been needed to get up to the 62,500 rate to begin with…

Therefore, it is strongly recommended that, if limit switch operation is to be used, these extremes be avoided. By default, the standard rate of change is initialized to 8000 microsteps/second/second, with the stop rate being set to 80 microsteps/second.

Also note that use of the "!" emergency reset command will cause an immediate stop of the motor, regardless of any other actions or settings in the system. ***Please be aware that, in some designs, damage to gear systems can result when such a sudden stop occurs. Use this feature with care!***

Note that as of version 1.47, it is possible to order the firmware configured for "instant stop" on the limit switches. As with the '!' command, if the firmware is configured with this mode of operation, ***please be aware that, in some designs, damage to gear systems can result when such a sudden stop occurs. Use this feature with care!***

Unused control signals: Y-,Y+,X-,X+,NX – Available as TTL Input signals

The input signalsY-(B0), Y+(B1), X-(B2), X+(B3) and NXT(B5) are not currently used by the NCStepper firmware. Note that they may be read using the "6?" report command, however.

To read the current values for these signals, you may send a request of

6?

The code will respond with:

R,6,nnn
*

Where 'nnn' is the value for the reading. It is bit-encoded as:

Bit	Value	Use
0	1	Y-
1	2	Y+
2	4	X-
3	8	X+
4	16	NXT
(5)	(32)	(RDY)
(6)	(64)	(SI)
(7)	(128)	(SO)

Only the low 5 bits are available as TTL input signals; the remaining are listed for completeness.

RDY/B6 Output Signal

The RDY output signal is suitable for running through a resistor/led pair (1K resistor, please) to indicate that motor motion is still being requested on at least one of the motors. When HIGH, then all motion is stopped. When LOW, at least one motor is still moving. This signal is LOW as long as there are pending actions in the line queue.

Serial Operation

The RS-232 based serial control of the system allows for full access to all internal features of the system. It operates at 2400 or 9600 baud, no parity, and 1 stop bit. Note that you should wait about ¼ second after power on or reset to send new commands to the controller; the system does some initialization processing which can cause it to miss serial characters during this "wake up" period.

Serial input either defines a new current value, or executes a command. The current value remains unchanged between most commands; therefore, the same value may often be sent to multiple commands, by merely specifying the value, then the list of commands. For example,

1000XY

would mean "Set the next locations of X=1000, Y=1000".

The firmware actually recognizes and responds each new command about ¼ of the way through the stop bit of the received character. This means that the command starts being processed about ¾ bit-intervals before completion of the character bit stream. In most designs, this will not be a problem; however, since all commands issue an '*' upon completion, and they can also (by default) issue a <CR><LF> pair before starting, it is quite possible to start receiving data pertaining to the command before the command has been fully sent! In microprocessor, non-buffering designs (such as with the Parallax, Inc.[tm] Basic Stamp [tm] series of boards), this can be a significant issue. The firmware handles this via a configurable option in the 'V' command. If enabled, the code will "send" a byte of no-data upon receipt of a new command character. This really means that the first data bit of a response to a command will not occur until at least 7-8 bit intervals after completion of transmission of the stop bit of that command (about 750 uSeconds at 9600 baud); for the Basic Stamp[tm] this is quite sufficient for it to switch from send mode to receive mode.

Selecting Baud Rate

By default, the baud rate on the system is fixed at 9600 baud, no parity, and 1 stop bit. Operation at 2400 baud may be selected during the initialization process via adding a resistor to ground to RDY. During reset (i.e., power on, hard reset, or processing of the "!" command), RDY is used as an input during reset to determine the baud rate. If it is tied to ground via a 1K resister, then the baud rate is set to 2400 baud. If it is left to float (the default), then the baud rate is set to 9600 baud.

Serial Commands

The serial commands for the system are described in the following sections. The code is case-insensitive (i.e., "s" means the same thing as "S"). ***Please be aware that any time any new input character is received, any pending output (such as the standard "*" response to a prior command, or the more complex output from a report) is cancelled.*** Additionally, for commands which have automatic waits built into them (such as "G" and "A"), the commands can be aborted, if more actions are still pending.

0-9, +, - – Generate a new VALUE as the parameter for all FOLLOWING commands

Possible combinations:

-n: Value is treated as -n

n: Value is treated as +n

+n: Value is treated as +n

Examples:

1000x – Set the next X value to 1000

-1000y – Set the next Y value to -1000

A – Draw an Arc of the requested radius

This command draws an "arc" (actually, a series of connected straight lines) whose radius is that specified, and whose other parameters were specified by the current state of the system. The complete arc is specified by:

- The X and Y commands set the CENTER point of the arc
- '**D**' defines the signed "delta angle per line", where the angle units are "degroids", each of which are 1/256 of a full circle (360/256 of a degree, or 1.40625 degrees)
- '**C**' defines the count of line segments; 0 means just draw a line from the current location of the length requested in the direction defined by the current '**B**'egin angle
- '**B**' specifies the beginning angle (again, in units of degroids, where one degroid = 1.40625 degrees)
- '**A**' specifies the radius of the circle, and actually draws the line

This command can take a very long time, since it operates by drawing the requested number of straight lines in a "circular" pattern. The "*" (ready) character is not sent back over the serial line until the last line segment has been queued to be drawn. If any new character except 'I' or '~' is received by the controller while drawing an "arc", then the arc drawing will be stopped at the next vertex. The 'I' character (see Idle Wait) may be used to see if the arc is still going; it immediately echoes back a status letter identifying the main action of the controller. The '~' character may be used as a "spacer" for communications timing – the 'A' and 'G' commands ignore it.

The firmware uses an internal table of sines to calculate the correct X,Y locations based on the center location and the current angle. The table provides scale factors accurate to about 1 part in 10,000,000; therefore, the actual coordinates calculated can be off by the greater of (1 part in 10,000,000) or (1 part in the radius). As long as the radius is less than 10,000,000, the values will be correct to within 1 unit of measure; otherwise, they will can be somewhat off (they are rounded to the nearest value based on the 1 part in 10,000,000 precision). They will be "perfect" when angle is at 0, 90, 180, or 270 degrees (0, 64, 128, or 192 "degroids").

Note that execution of the 'A' command will change the current values saved by the 'B' and 'C' commands, and will reset the current X and Y parameters to the last X,Y value requested as part of drawing the arc.

As a simple annotated example (the '*' is sent by the controller as its 'ready for next command' response),

*0x - Set X and
*0y - Y center point for the arc
*1d - Set the Arc-Delta to 1 degroid (1 degroid is 1/256 of a full circle; or 360/256 degrees)
*256c - Tell the system that there will be 256 steps to draw (256 small lines)
*0b - Begin "degroid angle" is 0
*1000a - Radius of Arc is 1000, and draw. This draws a "circle" of diameter 2000 steps.
*0x - Reset center back to 0,0 (otherwise, new center would be the last point drawn)
*0y
*256c - Reset count to 256; it gets destroyed with each draw
*0b - Reset arc angle; it is left at last point drawn
*2000a - Draw a 2000 unit radius circle
*0x - Once again, go to location 0,0 as center
*0y
*4c - This time, just set 4 lines in "arc"
*0b - Again, start at 0 "degroids"
*64d - set the unit delta to be 64; so that 4 will be a complete "circle"
*3000a - Draw the 3000 unit radius arc; since done in 4 steps, it is really a

square!
*

Note also that the 'A'rc command can be used to draw a line of a given length (within the limits of rounding and the 1:10,000,000 restriction, above) at any of the "degroid" angles from the current location. This can be done by specifying the 'B'egin angle as the desired value, the 'C'ount as 0, and the center X and Y values as the current location. For example,

*250x - Set X and
*300y - Y center point to the current location
*0c - Tell the system that there will be 0 steps to draw; we only go to the 'start' location
*32b - Begin "degroid angle" is 32 (which is 45 degrees)
*945a - Radius of Arc is 945, and draw. This draws a line of length 945 at a 45 degree angle
*

This allows you to easily move your motors to the real "start" position of an arc, by first doing a matching arc sequence with the count being 0. This greatly simplifies design of pen-plotter-like systems.

B – Select Beginning Arc Angle

'B' is used to select the starting angle (in 'degroids') for the 'Arc' command. See the 'A'rc command for more information.

After completion of the 'A'rc command, the 'B'egin angle is set to the last angle used in the drawing of the arc.

C – Define the arc Count of steps

'C' is used to define the number of line segments to draw as part of the 'Arc' command. A value of 0 causes the system to go just to the start point defined by the 'B'egin angle, the center X,Y, and the arc radius.

After completion of the 'A'rc command, the current 'C' value is undefined.

D – Define the arc Delta angle per step

'D' is used to define the count of "degroids" per arc step. This is the signed amount to add to the angle set by the 'B' command each time a new arc segment is drawn. The sign defines the direction of drawing: positive draws counter-clock wise (increasing angle), negative draws clockwise (decreasing angle).

G – Go to currently requested X, Y position; OR reset motor X,Y location to be the current X, Y parameter values.

This is normally used to queue the new X, Y location (from the X and Y commands) as the next location to target. Assuming that the "=" command has not been used to arm the goto as a "motor address reassignment" command, then the software will:

- Wait for the prior "pending" x, y location to actually be accepted to be drawn
- Calculate the direction and distance of travel for both motors, and the correct relative rates for the actions
- Queue the request to be started upon completion of the current motion
- Send back the "*" acknowledgement character

For example,

```
*1000X
*-25687Y
*g
```

Would:

1. Set the next X value to 1000
2. Set the next Y value to -25687
3. Queue a GOTO on motor location 1000, -25687

Note that the code will send back the "*" acknowledgement character as soon as the request has been queued; ***if there is a motion under way at present, and another already queued, the code will wait until a queue slot becomes available before it sends the '*' response***. If it receives another character while waiting for the slot, then the new GoTo action is aborted (i.e., the new X, Y location is never queued).

On the other hand, if the "=" command has been used to "arm" the "G" command to reset the motor location (instead of doing a real goto), then this command will wait for the motor to go completely idle, and then will reset the motor location to match the current X,Y parameter values. Once this has been completed, the code will reset the 'G' mode back to "GoTo", so that the next "G" command will actually do a goto. As a simple example here,

```
*2000X
*1000Y
*2=
*g
```

This would redefine the current location to be X=2000, Y=1000, with no real motion occurring. See the "=" command for more information about this special one-shot mode.

H – Operate motors at ½ power

"H" mode may be used to run a motor at a higher-than-rated voltage, in order to improve its torque. When 'H' is set to '1', then the PWM (Pulse Width Modified) count used to drive each winding is divided by two, thus cutting the effective current to the motor in half.

The two settings for this are:

0H – Run in normal FULL POWER mode (this is the power on/reset default)

1H – Run in ½ power mode

Note that if the "2W" mode is selected (for leaving windings on at ½ power when motion ceases), then the windings are actually left at ¼ power during idle. ***Please review the separate document "HalfPowerNotes.pdf" for a complete description of correct use of this capability.***

Note that, for firmware versions before 1.42, the board reverts to mode "0H" (full power mode) whenever it is reset (by a power cycle, low pulse on the RESET input line, or via the "!" command). If your code fails to detect this reset condition, you can cause a board failure by not re-issuing the "1h" command after a reset! Starting with firmware version 1.42, the code can automatically select the initial state of the ½ power mode by sensing the presence of a 1K resistor between the SO (Serial Output) signal and ground. If there is no resistor there, then full power mode will be selected (i.e., the same behavior as prior code versions). However, if there is a 1K resistor between SO and GND, then the firmware will select ½ power mode as the initial state, thus avoiding potential damage to the controller.

I – Wait for motor 'Idle'

This allows your code to 'wait' for the motors to be idle. It also immediately reports what main type of action is occurring; this allows your code to confirm whether a new command can be sent. Sending an 'I' is always safe (even if you have not received an '*' from another command); this allows you to easily 'poll' the board in a multi-port environment.

Most commands are executed immediately, and are not affected by new incoming data. However, both "G" (goto) and "A" (draw arc) have the restriction of not being able to queue more than 2 vectors (the one being drawn, and one which is pending). If a character is received while either of those commands are waiting for "room" to queue their next vector request, then the command ("G" or "A") which is waiting to be queued will be aborted. Neither of those commands will send the "*" (ready for new command) character until they have queued their last request; however, in a multi-board environment, your application may miss the "*". The "I"dle wait command gives you a method of safely resynchronizing with the board – it immediately sends back the type of wait which it is doing (as a single character response after the optional CR/LF command acknowledgement), and then will send the "*" when the command completes or is queued, as is appropriate.

The code immediately sends back one of 3 characters to inform you about the pending board operation:

- 'A' – The board is currently waiting on execution of an "arc" (multi-line segment) request
- 'G' – The board is currently waiting for a queue slot to enqueue a new goto target
- 'I' – the board has no Arc or unenqueued GoTo pending

After sending the above status character, the code then waits as is appropriate (depending on whether the unit is waiting on enquing an 'A' or 'G' command), and then sends an '*' response.

- Waiting on 'A': The '*' is sent as soon as the last vector of the arc is queued (motor motion is still occurring)
- Waiting on 'G': The '*' is sent as soon as the 'G' is queued (motor motion is still occurring)
- Waiting on 'I'dle: The '*' is sent once all motor motion has completed; the motors are stopped.

If you send ***any*** new command other than 'i' during the 'A' or 'G' styles of idle wait, then the 'A' will be aborted (i.e., the currently enqueued vectors will execute, but no further ones will be added to the list), and the 'G' will be discarded.

Additionally, if the wait is stopped by receipt of a new character, then the new character IS processed as part of a new command – it is NOT discarded.

For example, to go to a given X, Y location, and then wait completely for the motor to actually get there, you could simply issue the command sequence:

Send	Receive
1000X	*
3000Y	*
G	* *(note that the '*' is received as soon as the new X, Y location is actually accepted as the next item which will be targeted)*
I	I* *(note that this '*' is not received until all motion completes)*

If you already had multiple "G" requests queued, and you issued the "I" before the "G" was able to queue its request, then you might have received a "G" instead of an "I" in response to the "I" request, above. This would mean that "G" is still trying to queue its request, and that it is not safe to send any other character except 'I'. The "G*" response would be sent if the board has just enqueued the G command. The "I*" response would be sent if the board is truly idle.

K –Set the "Stop oK" rate

This defines the rate at which the motors are considered to be "stopped" for the purposes of stopping or reversing directions. It defaults to the default of '80' if a value of 0 is given.

By default, this is preset to "80" upon startup of the system. This means that, whenever a stop is requested, the motor will be treated as "stopped" when its stepping rate is <= 80 microsteps (5 full steps) per second.

For example,

100k

sets the stop rates for the currently selected motor(s) to be 100 microsteps per second. Any time the current rate is less than or equal to 100, the motor will have the ability to stop instantly.

To set the rate such that the motors always immediately start and stop at the desired rate ('R') setting, issue the command:

62500K

This sets the 'Stop oK' rate to the maximum possible step rate, and thus will prevent all ramping behaviors of the code.

This action automatically waits for all motor motion to complete before executing. It will not send back its '*' response until the motors are completely idle. If a new (non-'I') character is received before this happens, then the 'K' command is forgotten.

L – Latch Report: Report current latches, reset latches to 0

The "L"atch report allows capture of key short-term states, which may affect external program logic. It reports the "latched" values of system events, using a binary-encoded method. Once it has reported a given "event", it resets the latch for that event to 0, so that a new "L" command will only report new events since the last "L".

The latched events reported are as follows:

Bit	Value	Description
0	+1	Y- limit reached during a Y- step action
1	+2	Y+ limit reached during a Y+ step action
2	+4	X- limit reached during a X- step action
3	+8	X+ limit reached during a X+ step action
4	+16	System power-on or reset ("!") has occurred
5	+32	'G' or 'A' command was probably aborted – a non "~" or "I" character was received while the G or A was waiting for queue space to save a vertex location.

For example, after initial power on,

```
L
```

Would report

```
L16
*
```

If you were then to do an X seek in the "-" direction, and you hit an X limit, then the next "L" command could report:

```
L4
*
```

O – step mOde – How to update the motor windings

The windings of the motors can be updated in one of three ways, depending on this step mode setting. By default, the code uses "micro step" mode set for 8 steps per complete full step, and performs a near-constant-torque calculation for positions between full step locations. The other modes include two full step modes and an alternating mode. For the full step modes, one enables only 1 winding at a time (low power), while the other enables 2 windings at a time (full power). The remaining mode alternates between 1 and 2 windings enabled.

The values which control this feature are:

- 0 : Full Step, Single winding mode (1/2 power full steps)
- 1 : Half step mode (alternate single/double windings on – non constant torque)
- 2 : Full step, double winding mode (full power full steps)
- **3 : Microstep, as fine as 1/8th step, constant-torque mode – This is the power on/reset default stepping mode.**

For example,

0o

sets the above ½ power full step mode, while

3o

sets the default microstep mode.

The "o" command does NOT affect the current step rates or locations; it only affects how the windings are updated. For example, when operating in the 1/8th step size, the following rules are applied for the various modes.

- 0: Single winding full step mode: Exactly one winding will be on at a time, and will be on at the selected current for the motor. The "real" physical motor position (in full step units) therefore only updates once every 8 microsteps; thus the "full step" location will be the (microstep location)/8, dropping the fractional part.
- 1: Half step mode: Alternates between having one and two windings on at a time, thus causing the torque to vary at the half-step locations. The "real" physical locations will be at half-step values, and hence the motor will "move" once every 3 microsteps. The "full step" location will be the (microstep location)/8, with fractions of 0 to 3/8 mapping into fractional location 0, and 4/8 though 7/8 mapping into fractional location 0.5.
- 2: Double winding full step mode: Both windings are "on" (at the selected motor current) at a time. As with mode 0, the "real" physical motor position will actually only update once every 8 microsteps. The "full step" location will be the (microstep location)/8, with the fractional part forced to 0.5.
- 3: Microstep mode. The current through the windings are precision-controlled, so that the microposition can be obtained. The physical motor position expressed in full step units is the (microstep location/8).

P – sloPe (number of steps/second that rate may change)

This command defines the maximum rate at which the selected motor's speed is increased and decreased. By providing a "slope", the system allows items which are connected to the motor to not be "jerked" suddenly, either on stopping or starting. In some circumstances, the top speed at which the motor will run will be increased by this capability; in all cases, stress will be lower on gear systems and motor assemblies.

The slope can be specified to be from 1 through 62,500 microsteps per second per second. If a value of 0 is specified, the code forces it to have a value of 8000. If a value above

62,500 (or less than 0) is specified, the code will accept it, but will ramp unreliably (i.e., do not do it!).

This value defaults at power-on or reset to 8000 microsteps per second per second. Please note that changing this during a "goto" action will cause the stop at the end of the goto to potentially be too sudden or too slow – it is better to first stop any "goto" in progress, and then change this slope rate.

For example, if we are currently at location (0,0) then the sequence:

250p500r0X2000Yg

would cause the following actual ramp behaviors to occur:

1. The motors would start at their "stop oK" rate, such as 80 microsteps/second
2. The Y motor would accelerate to its target rate of 500 microsteps per second, at an acceleration rate of 250 microsteps/second/second.
3. This phase would last for approximately 500/250 or about 2 seconds, and would cover about 500 microsteps of distance.
4. It would then stay at the 500 microstep per second target rate until it was about 500 microsteps from its target location, i.e., at location 1500 (which would take another 2 seconds of time).
5. It would then slow down, again at a rate of 250 microsteps per second, until it reached the stop oK rate. As with the acceleration phase, this would take about 2 seconds.
6. The total distance traveled would be exactly 2000 microsteps, and the time would be 2+2+2=6 seconds (actually, very slightly less).

This action automatically waits for all motor motion to complete before executing. It will not send back its '*' response until the motors are completely idle. If a new (non-'I') character is received before this happens, then the 'P' command is forgotten.

R – Set run Rate target speed for the faster motor

This defines the run-rate to be used for the faster motor. It may be specified to be between 1 and 62,500 microsteps per second. If a value of 0 is specified, the code forces a value of 400. If a value outside of the limits is specified, then it is accepted, but the code will not operate reliably. As with the ramP rate, do not specify values outside of the 1-62,500 legal domain.

This defines the equivalent number of microsteps/second which are to be used to run the faster motor under the GoTo or Arc command. The internal motor position is updated at this rate, using a sampling interval of 62,500 update tests per second. The motor windings are then updated according to the stepping mode. For example, if the stepping mode (the 'o' command) for a given motor is one of the full-step modes instead of the microstep mode, and the microstep resolution is set to '1', then the motor will actually experience motion at 1/64th of the specified rate.

For example,

250R

Sets the stepping rate to 250 microsteps per second.

The power-on/reset default Rate is 800 microsteps/second.

This action automatically waits for all motor motion to complete before executing. It will not send back its '*' response until the motors are completely idle. If a new (non-'I') character is received before this happens, then the 'R' command is forgotten.

V – Verbose mode command synchronization

The 'V'erbose command is used to control whether the board transmits a "<CR><LF>" sequence before it processes a command, and whether a spacing delay is needed before any command response. **By default (after power on and after any reset action), the board is configured to echo a carriage-return, line-feed sequence to the host as soon as it recognizes that an incoming character is not part of a numeric value**. This allows host code to fully recognize that a command is being processed; receipt of the <LF> tells it that the command has started, while receipt of the final "*" states that the command has completed processing.

The firmware actually recognizes and responds each new command about ½ of the way through the stop bit of the received character. This means that the command starts being processed about ½ bit-interval before completion of the character bit stream. In most designs, this will not be a problem; however, since all commands issue an '*' upon completion, and they can also (by default) issue a <CR><LF> pair before starting, it is quite possible to start receiving data pertaining to the command before the command has been fully sent! In microprocessor, non-buffering designs (such as with the Parallax, Inc.^{tm} Basic Stamp ^{tm} series of boards), this can be a significant issue. This is handled via a configurable option in the 'V' command. If enabled, the code will "send" a byte of no-data upon receipt of a new command character. This really means that the first data bit of a response to a command will not occur until at least 9 bit intervals after completion of transmission of the stop bit of that command (about 900 uSeconds at 9600 baud); for the Basic Stamp^{tm} this is quite sufficient for it to switch from send mode to receive mode. *The Firmware also adds 2 additional "stop" bits to each transmitted character, when this feature is enabled. This is to allow non-buffering microprocessors some additional time to do real-time input processing of the data.*

The verbose command is bit-encoded as follows:

Bit	SumValue	Use When Set
0	+1	Send <CR><LF> at start of processing a new command
1	+2	Delay about 1 character time before transmission of first character of any

		command response. Add 2 more stop bits to each transmitted character, to allow more processing time in the receiving microprocessor.

If you set verbose mode to 0, then the <CR><LF> sequence is not sent. Reports still will have their embedded <CR><LF> between lines of responses; however, the initial <CR><LF> which states that the command has started processing will not occur.

For example,

0v

would block transmission of the <CR><LF> command synch, and could respond before completion of the last bit of the command, while

3v

would enable transmission of the <CR><LF>sequence, preceded by a 1-character delay. The complete table of options is:

Value	Delay First	<CR><LF>
0	No	No
1	No	Yes
2	Yes	No
3	Yes	Yes

W – Set windings power levels on/off mode

The "W"indings command controls whether the motors have their windings left enabled or disabled once any GoTo or Slew action has completed, and it controls power levels to use during normal stepping. It is acted on immediately – that is to say, if the current motor(s) is (are) stopped, then the windings are *immediately* engaged or disengaged as requested.

The values to use for control are:

0w – Full power during steps, completely off when stepping completed. This is the power-on/reset default condition.

1w – Full power at all times (both during steps and when idle)

2w – Full power during steps, 50% power when idle

This mode is used to reduce power consumption for the system. When windings are disengaged, they draw very little power; however, their full rated power is drawn when they are engaged. If windings are off, then the stepper motor will "relax", and will move on its own to a "preferred location", controlled by its fixed magnets (thus inducing up to ½ step's worth of positional error). If they are on, the motor is actively held at its requested location (and the motor itself heats up). If mode 2 is used (the 50% power setting), then the windings are pulsed at about ½ of the normal rate, thus the power requirements are ½ of the normal amount for the given location, after a goto or slew has completed.

X – Set next X value as defined by the current "=" mode

This command sets the next X parameter to the value requested, as defined by the current "=" mode. By default, this is a direct assignment of the value specified to the pending X register.

For example,

100X

Would normally set the next X value to be 100.

However, if the "=" command has been used to reset the default assignment mode from absolute to relative, then each "X" command will simply do a signed addition of the new value to the prior X, Y request value. For example

100X200X

would start by setting X to 100 (assuming it 'really' started at 0), and then would add 200 to that, thus making the current X be 300.

Y – Set next "Y" value as defined by the current "=" mode

This command sets the next Y parameter to the value requested, as defined by the current "=" mode. By default, this is a direct assignment of the value specified to the pending Y register.

For example,

100Y

Would normally set the next Y value to be 100.

However, if the "=" command has been used to reset the default assignment mode from absolute to relative, then each "Y" command will simply do a signed addition of the new value to the prior X, Y request value. For example

100Y-200Y

would start by setting Y to 100 (assuming it 'really' started at 0), and then would add -200 to that, thus making the current Y be -100.

Z – Stop motors.

'Z' causes the motors to be ramped to a complete stop, according to the current ramp rate and stepping rate. "Stopped" is defined as "having a step rate which is <= the stop oK rate". See the 'K' command for defining the "stop oK rate".

! – RESET – all values cleared, all motors set to "free", redefine microstep. Duplicates Power-On Conditions!

This command acts like a power-on reset. It ***IMMEDIATELY*** stops both motors, and clears all values back to their power on defaults. No ramping of any form is done – the stop is immediate, and the motors are left in their "windings disabled" state. This can be used as an emergency stop, although all location information will be lost.

The value passed is used as the new microstep size, in fixed $1/64^{th}$ of a full step units. **At raw power on, the board acts like a "4!" has been requested**; that is to say, it sets the microstep size to 4x1/64, which is $1/16^{th}$ of a full step. By issuing the '!' command, you can redefine the microstep size to a value convenient for your application. The value must range from 1 to 64; it is clipped to this range if exceeded.

The suggested values would be the powers of 2, vis. 1, 2, 4, 8, 16, 32 and 64 (giving you true microstep step sizes of 1/64, 1/32, 1/16, 1/8, ¼, ½ and 1, respectively). All other values (such as RATE or GOTO LOCATION) are then expressed in units of the microstep size; therefore, location "3" would mean "3/64" in the finest resolution (microstep set to 1), and "3" in the largest resolution (microstep set to 64). Note: versions of firmware prior to 1.43 allowed up to ½ step as being the largest microstep size (i.e., a value of 32 for this function).

For example,

```
4!
```

resets the system to its power on default of 1/16 microstep resolution.

The reset command also selects the following settings:

- **2=** – Select the next "G"oto assigns the location instead of moving the motors
- **0X** – Set next X will be 0
- **0Y** – Set next Y will be 0
- **G** – Reset both motors to be at location 0
- **0H** – Set motors to full power mode
- **80K** – Set the "Stop OK" rate to 80 microsteps/second
- **3O** – Set the motor windings Order to "microstep"
- **8000P** – Set the rate of changing the motor speed to 8000 microsteps/second/second
- **800R** – Set the target run rate for the faster motor to 800 microsteps/second
- **1V** – Set <CR><LF> sent at start of new command, no transmission delay time
- **0W** – Full power to motor windings

= – *Define interpretation of "X", "Y", and "G" commands*

This command is used to define how the X and Y commands operate (that is to say, whether they directly assign values to the X and Y parameter variables, or they add new offsets to those values), and to define whether the "G" command actual operates as a "GoTo" or as a one-shot "assign location" command.

It is bit encoded as follows:

Bit	Use
0	0 = Assign X, Y to value requested
	1 = Add value requested to X, Y (i.e., 1000X would add 1000 to the current pending X value)
1	0 = "G" queues the current X, Y parameters as a new GoTo target
	1 = The next "G" waits for all motor motion to stop, then assigns the current motor location from the current X, Y values. Note that "G" then automatically clears this bit after performing the assign

Therefore, the legal values become:

Value	Use
0	Default: "X" and "Y" commands are absolute, "G" is a "GOTO"
1	"X" and "Y" commands are relative to their current values, "G" is a "GOTO"
2	"X" and "Y" commands are absolute, the next "G" command is an "ASSIGN MOTOR LOCATION"
3	"X" and "Y" commands are relative to their current values, the next "G" command is an "ASSIGN MOTOR LOCATION"

As an annotated example (the '*' are sent by the controller),

```
*2=     - Set the X and Y parameter mode to absolute, next G assign location
*0X     - Set current X and
*0Y     - Y parameters to 0
*G      - Since we were in "Assign motor location" mode, the current location
              is now 0,0; no actual motor motion occurs
*1=     - Set the X and Y parameter mode to relative
*200X   - X parameter is now 0+200➔200
*-300Y  - Y parameter is now 0-300➔-300
*G      - go to location 200,-300
*-100X  - X parameter is now 200-100➔100
*200Y   - Y parameter is now –300+200➔-100
*G      - go to location 100,-100
```

? – Report status

The "Report Status" command ("?") can be used to extract detailed information about the status of either motor, or about internal states of the software.

For a status report, the value is interpreted as from one of three groups:

1-255: Report memory location 1-255

Useful locations:

- 5: Port A register – this contains the limit switches
- 6: Port B register – this contains the TTL inputs
- 7: Port C register – this controls the motor windings

0: Report all of the following special reports except for version/copyright

-1 to -12: Do selected one of the following reports

- -1; Report current X location
- -2; Report current Y location
- -3; Report target X location
- -4; Report target Y location
- -12; Report current software version and copyright
- other: Treat as 0 (report all except version/copyright)

All of the reports follow a common format, of:

1. If Verbose Mode is on, then a <carriage return><line feed> ("crlf") pair is sent.
2. 'R' is sent.
3. A comma is sent.
4. The report number is sent (such as –4, for target position).
5. Another comma is sent.
6. The requested value (or set of values) is (are) reported.
7. If Verbose Mode is on, then a <crlf> is sent

Finally, a "*" character is sent, which notifies the caller that the report is complete.

The special reports which are understood are as follows:

0: Report all reportable items

The "report all reportable items" mode reports the data as a comma separated list of values, for reports –1 through –4. Just after power on, for example, the request of "0?" would generate the report:

R,0,a,b,c,d

Where:

- R is the "Report" flag character
- 0 is the report number; 0 is the 'all' report
- a is the value for the current X location (report "-1")
- b is the value for the current Y location (report "-2")
- c is the value for the target X location (report "-3")
- d is the value for the target Y location (report "-4")

For example,

```
B0?
```

Would report all reportable values for both motors. You could receive:

```
X,0,30,10,1000,30,10
*
```

-1: Report current X location

This reports the current (instantaneous) location for the X motor.

For example,

```
-1?
```

You could receive:

```
R,-1,10
*
```

-2: Report current Y location

This reports the current (instantaneous) location for the Y motor.

For example,

```
-2?
```

You could receive:

```
R,-2,2502
*
```

-3: Report the target X location

This reports the current GoTo X target location.

For example,

```
-3?
```

You could receive:

```
R,-3,10
*
```

-4: Report the target Y location

This reports current GoTo Y target location.

For example,

```
-4?
```

You could receive:

```
R,-4,-35443
*
```

-12: Report current software version and copyright

This reports the software version and copyright.

For example,

```
-12?
```

could report:

```
NCStepper.src $version: 1.29$
©2002 by Peter Norberg Consulting, Inc. All Rights Reserved
*
```

Any character above 'z' – stop sending pending output data, then skip

Characters above 'z' (such as '{', '~', and '}') are actively ignored by the processing code for most actions. Any pending output (such as the copyright message or any prior responses) will be cancelled, the current numeric value will normally be treated as completed, and the character will otherwise be fully ignored.

We suggest that the '~' character be used by applications which are not maintaining a perfect synch with the board as a "clear output buffers". For example: Send the sequence

~~I

Then start to monitor the incoming data stream for new serial characters received since transmission of the "I" (i.e., discard all data received prior to the 'I' being transmitted). That data will strictly be the response to the 'I' (idle wait) command; there will be no "left-overs" from prior commands.

other – stop sending output data, then echo the '' command complete response*

Any illegal command is normally ignored, other than sending a response of "*". However, if a numeric input was under way, that value will be treated as complete. If an "A" or "G" command still has pending actions waiting to be queued, the queued actions are aborted.

For example,

123 456X

would actually request a "Set X parameter to 456". Since the " " command is illegal, it is ignored; however, it terminates interpretation of the number which had been started as 123.

Note that, upon completion of ANY command (including the 'ignored' commands), the board sends the <carriage return><line feed> pair, followed by the "*" character.

SerTest.exe – Command line control of stepper motors

The SerTest.exe application (provided as part of the sample software) is a simple tool which allows command line based control of the NCStepper product line (the SimStep and BiStep boards). It allows a batch-based script to control stepper motors, with no further need for any programming knowledge. All sources are provided, to allow rewriting as needed.

SerTest allows you to send command strings and see their responses, by issuing commands from the command prompt window. It is called as

```
SerTest Text1 Text2 … Textn
```

Where "Text1", "Text2", … are the actual strings to send to the controller (as described in the "Serial Commands" section of this manual). The code supports extended control of its behavior, by parsing the first character of each space-separated parameter on the command line – if it starts with "/", then the rest of that parameter is interpreted as a command to SerTest, instead of being sent to the controller. The commands recognized by SerTest are:

- /b#### Set Baud rate to ####; defaults to /b9600

  For example,

  /b9600 sets 9600 baud,

  /b2400 sets 2400 baud. No other values are useful.

- /c??? Set the extended list of "long wait" commands

  This is used to set the list of commands which may take a long time to respond. By default, just "I" (for "Idle Wait") is specified to take more than $1/10^{th}$ of a second; however, when operating the NCStepper firmware, the commands "A" (for "draw Arc") and "G" (for "Go draw the current line) can take a large amount of time, if no queue space is left for saving the new requests. When using the NCStepper firmware, you therefore should include the switch:

  /cag

  in order to add the "A" and "G" commands to the idle wait time list.

- /i#### Set Idle wait time to #### milliseconds; defaults to /i60000

  The "Idle wait time" is the maximum amount of time (in milliseconds) which the software waits before it decides that a command has timed out, and thus that it is time to send the next command. This is used to maintain correct synchronization of the code with the controller. For example,

  /i60000 – Set 1 minute before timeout

  /i10000 – set 10 seconds before timeout

- /pCOMn set the serial communications port to port n; defaults to /pCOM1

  This allows control of which serial port is used for the following commands. The code does not actually attempt open any serial port until the first real data is ready to be sent to the controller; thus no attempt will be made to access COM1 if the command line looks like:

  SerTest /pCOM2 4!x1000g

  Note that if multiple /p commands are on the line, the most recent one seen is the one used at any given time. It is legal to have one command line actually operate multiple controllers!

All other text is passed, unchanged, to the controller. SerTest is aware of the general command structure for the StepperBoard product line; thus, it will correctly wait for synchronization each time a complete command is sent. All data received by SerTest is echoed back to the command prompt, thus allowing the operator to see the response to any command (or set of commands).

For example,

Sertest /cag 4!1000x-2000ygi

Would:

1. Operate at 9600 baud on COM1 using a 1 minute time out
2. reset the board to operate with a microstep size of 4/64
3. tell the X motor to go to location 1000,
4. tell the y motor to go to location -2000,
5. and wait up to 60 seconds for the motions to complete

Similarly,

SerTest /cag /pCOM2 /b2400 /i10000 5000yg

Would:

1. Operate using port COM2 at 2400 baud, with a timeout of 10 seconds
2. Tell the Y motor to seek to location 5000

StepperBoard.dll – An ActiveX controller for StepperBoard products

The StepperBoard.dll object is a fairly comprehensive sample Visual Basic COM/ActiveX application which allows any COM-aware system (such as VBScript based scripts) to easily control the StepperBoard products. As with the SerTest application, all sources are provided, so that the user may change the system as needed.

The program is well-documented in the manual StepperBoardClass.pdf. Please refer to that manual for more information about the product.

Calculating Current And Voltage Power Supply Requirements

This section note describes how to calculate the power requirements for your motors, and for the system as a whole.

1. Determine the individual motor winding current requirements.

The first issue is to determine the individual winding current requirements for your stepper motor. Since our system does not monitor current at all (it only estimates current, using a PWM-like technique), the current ratings as seen by our board may not match those specified by a manufacturer who is assuming that current-monitoring based control is being performed.

From the point of view of determining the current requirements for your motor, our system is best modeled using the standard resistor-only based formula (ignoring inductance) of:

V=IR

or, rearranging terms in order to find I,

I=V/R

That is to say, the current (I) as seen by our board equals the voltage (V) from your power supply divided by the resistance (R) of your motor windings. This value can be much greater than that claimed by a given motor manufacturer, since most of them assume that you are using a current-controlled system to run their motors.

For example, if you have a 3 ohm resistance in your windings, then the motor will "draw" 6/3 or 2 amps if 6 volts is driven out of it, and it will draw 12/3 or 4 amps (per winding!) if 12 volts is generated.

2. Determine current requirement for actually operating the motor(s)

Once you have determined the motor current, then you will need to determine how you intend to run it via our product offerings. We have four modes of operation, which provide for three levels of power per motor. These modes are controlled by the "o" command, which specifies the technique used to drive the windings.

Update Order	***Absolute Current Multiplier***	***Recommended Current Multiplier***
0 (single winding full step)	1.0	1.4
1 (half step: alternate 1, 2 windings)	2.0	2.5
2 (full step: 2 windings at a time)	2.0	2.5
3 (microstep)	1.7	2.3

Note that the "Recommended Current Multiplier" column in the above table includes a "fudge factor"; we always recommend using a power supply which is somewhat larger than the absolute minimum required, in order to avoid overloading issues.

Obviously, if you are going to run multiple motors off of one supply, you will need to add together all of the currents needed in order to determine how large of a supply to use.

For example, if you are going to microstep (mode 3) a motor whose winding current has been calculated to be 0.4 amps, then your power supply needs to be able to supply 2 x 0.4, or 0.8 amps to drive that particular motor.

3. Determine the voltage for your motor power supply

No, this is not "who is buried in Grant's Tomb". With the exception of the SimStepA04 and SS0705 units, all of our drivers lose some of the voltage from the power supply before delivering it to the motor. The amount of loss depends on the controller, the current being supplied as determined by the above I=V/R formula, whether the motor is being operated using our 'double current mode', and whether the motor is being connected to the controller as a Unipolar or Bipolar drive based system.

The approximate voltage drop for a bipolar motor driven by the BiStep series of products operating in both standard current mode and double current mode can be estimated from the following table: please note that these are only estimates, and will vary somewhat based on actual components, motors, and temperature. A unipolar motor will cause a voltage drop of approximately 1/2 of the amount shown.

Standard Current Mode, Single Winding Current	***Double Current mode, Single Winding Current***	***Approximate Voltage Drop for a BiPolar Motor***
0.5	1.0	Up to 2 volts
1.0	2.0	Up to 3 volts
2.0	4.0	Up to 5 volts

For example, from the above table we would estimate that at most a 15 volts supply should be used to drive a 12 ohm bipolar motor at 1 amp of current when operating in the standard (double motor) mode of operation. Probably, for safety, we should use a supply slightly less than that if any unit other than the BiStep2A is used as the controller, since this is right at the limits of the BiStepA06 system.

4. Determine the logic supply requirements

The current needed by the logic portion of our product offerings depend upon the product. All of the units except for the BiStep2A need at most 0.4 Amps for their logic (that provided via the Vc or Va connection); the BiStep2A requires 1 amp for this connection.

Note that the logic supply (+Vc) must always be in the range of 7.5 to 15 volts. If less than 7.5 volts is used, the regulator will not operate reliably (causing the board to reset itself, losing motor control and position information). If greater than 15 volts is used, you are likely to 'blow out' the logic voltage regulator, damaging the board.

5. Determine the power supplies you will be using

Your choices are dependent on the desired voltage to the motors, and on the board which you have purchased from us. In all cases, we strongly recommend that linear supplies be used: switching supplies are not very good when used with inductance based loads.

The logic supply must be in the voltage range of 7.5 through 15 volts. If your motor voltage requirements are outside of this range, then you will have to use a split supply (as described below).

Single Supply.

If your motor power supply voltage is from 7.5 to 15 volts, then you may choose to use a single supply to operate the system (on the BiStepA04, you may only use a single supply; therefore, you must use a supply which is in this voltage range). Obviously, the current capabilities of the supply must exceed the sum of the current requirements of the motor(s) and the logic circuits.

Dual Supply

For all units **except the BiStepA04**, you may separate the motor supply from the logic supply. If you do so, we suggest using the lowest voltage in the range of 7.5 to 15 volts on the logic supply which you have available, to reduce generation of waste heat on the board.

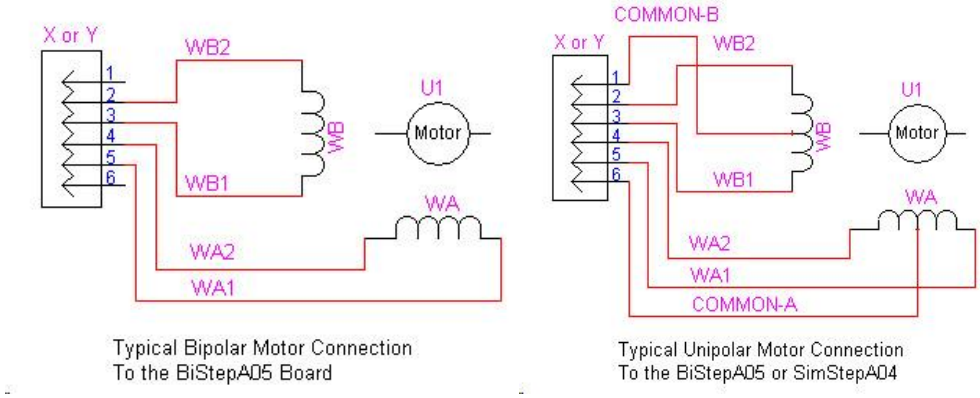
The motor supply should be above 5 volts in all cases (due to some signal requirements on the board), and otherwise is as calculated under sections 1 through 3, above. If the supply is to drive 2 motors, please remember to double the current needs.

Triple Supply

The BiStep2A has the additional capability of allowing use of three different power supplies. It is quite possible to have one motor operate at a different voltage level than the other, if this turns out to be a requirement.

Wiring Your Motor

There are two identical connectors used to operate the X and Y motors. The connectors are labeled with respect to which motor they operate. (This designation affects only which commands are to be used to control the motors; no other functionality is changed.) They are wired as follows for the SimStepA04, SS0705, BiStepA05, BiStepA06 and the BiStep2A series of controllers (pins counting from top to bottom):



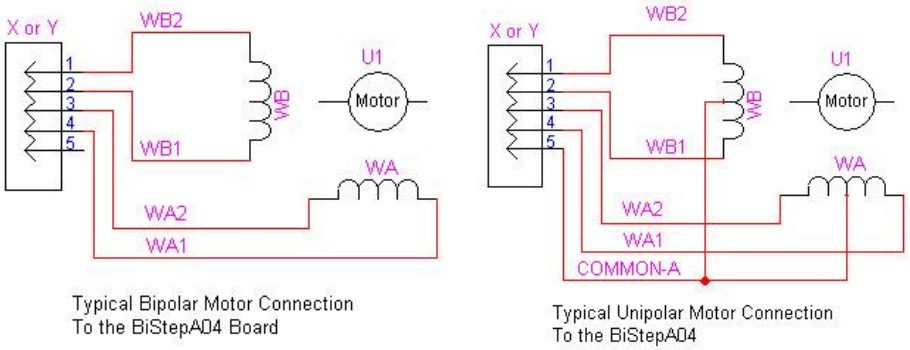
Typical Bipolar Motor Connection
To the BiStepA05 Board

Typical Unipolar Motor Connection
To the BiStepA05 or SimStepA04

Pin	Name	Description
1	GND or +V	If BiStep board, GND; if SimStep or SS0705 board, +V
2	WB2	Winding B, pin 2
3	WB1	Winding B, pin 1
4	WA2	Winding A, pin 2
5	WA1	Winding A, pin 1
6	GND or +V	If BiStep board, GND; if SimStep or SS0705 board, +V

This pinout was selected to allow simple reversing of the connector (i.e., take it out and turn it around) to reverse the direction of the motor if a non-polarized connector is used.

The BiStep A04 board does ***not*** support the above reversing functionality. The "top" ground pin is removed (the connectors are 5 pins instead of 6), generating the pinout:



Typical Bipolar Motor Connection
To the BiStepA04 Board

Typical Unipolar Motor Connection
To the BiStepA04

Pin	Name	Description
1	WB2	Winding B, pin 2
2	WB1	Winding B, pin 1
3	WA2	Winding A, pin 2
4	WA1	Winding A, pin 1
5	GND	Ground return

Stepping sequence, testing your connection

The current is run through these connectors to generate a clockwise sequence as follows:

Step	***WB2***	***WB1***	***WA2***	***WA1***
0	0	0	0	**1**
1	0	**1**	0	**1**
2	0	**1**	0	0
3	0	**1**	**1**	0
4	0	0	**1**	0
5	**1**	0	**1**	0
6	**1**	0	0	0
7	**1**	0	0	**1**

Note also that it is explicitly legal when using the GenStepper firmware to operate your motor in "double current, ½ power mode". In "double current" mode, you wire your motor to both the X and Y motor connectors, and you jumper the "LY-" and "LY+" signals to ground to tell the board that double current mode is enabled. In all other respects, you follow the rest of the instructions in this manual.

The actual wiring configuration to connect to a given stepper motor depends on the motor type. For most unipolar motors, each winding has three leads. The center-tap (shown in the above schematics as "COMMON-A" or "COMMON-B") is connected to the GND signal in the BiStep series controllers, or to +Vm on the SimStep/SS0705 series of controllers. The other two leads are connected to pins WA-1 and WA-2 or WB-1 and WB-2, as shown in the above schematics. For bipolar motors, the windings match the labels – that is to say, pins 2-3 are for winding B, and 4-5 are for winding A. Note that the unipolar motors will also match the labels, but it may be more difficult to identify the windings.

Determining Lead Winding Wire Pairs

If there is no manufacturer's wiring diagram available, unipolar and bipolar motor windings can both often be identified with an ohm-meter by performing tests of their resistances between the motor leads.

For any motor, number the leads (from 1 to 4 for a bipolar motor, from 1 to 5 or 6 for a unipolar motor). Then measure the resistances and record the values in the empty cells in a table like the following:

	1	**2**	**3**	**4**	**5**	**6**
1	-					
2	-	-				
3	-	-	-			
4	-	-	-	-		
5	-	-	-	-	-	
6	-	-	-	-	-	-

For example, the cell at location (1,2) would be filled in with the resistance between leads 1 and 2. The '-' entries show values which do not need to be separately measured, since they are already measured in another row/column pair (or are a self-reading). For example, having measured the resistance between leads 1 and 2 to fill in cell (1,2), there is no reason to separately measure leads 2 and 1! If you have fewer leads than those shown in the table, ignore the rows and columns with the nonexistent leads.

For a 4-wire bipolar motor, the low-resistance pairs are the opposite ends of matching windings; high-resistance pairs are different windings. For example, if cell (1,2) shows 10 ohms, while (1,3) shows greater than 1000 ohms, then wires 1 and 2 can be called winding A, while wires 3 and 4 can be called winding B.

For a 5-wire unipolar motor, you will observe 2 reading values in the resulting table, with the higher reading being about double that of the lower reading. The single line which has the lower reading on all of its entries in the table is the common lead; the other wires are the winding leads (unfortunately, this test cannot show which is winding A and which is winding B through resistances alone).

For a 6-wire unipolar motor, you will observe 3 reading values in the resulting table.

- If you see a single reading near 0, then the two leads associated with that reading are the common leads, and the remaining 4 wires are the windings WA1, WA2, WB1 and WB2 (this test cannot determine which is winding A or B through resistances alone). As a check, you can observe that all readings between the other wires and either of the 2 common wires have value ½ that of all of the readings between the non-common wires.
- Otherwise, you will see readings which are near infinity (which identify leads from different windings), are at some value (such as 10), or are at double that value (such as 20). The pairs which show the "double value" are the opposite ends of a given winding (i.e., WA1 and WA2, or WB1 and WB2). The remaining wires are the "common" leads for their given windings.

A 6-wire 4-phase unipolar motor will have two "common" wires. You will normally connect one of the wires to pin 1, and the other to pin 6. However, you can often operate a 6-wire unipolar motor as if it were a 4-wire bipolar motor (when using the BiStep series of controllers) by insulating the common leads and leaving them disconnected. When it works, this usually provides more torque for the motor, but it requires double the voltage (at the same level of current) from the power supply. You cannot operate with this pair of wires disconnected if they are connected together inside the stepper motor -- if the resistance between the common leads is very low (less than 10 ohms), such a connection exists and you must therefore operate using the regular unipolar wiring scheme.

Sequence Testing

Always double check all of your power and motor connections before you apply power to the system. If you have reversed any power leads, you will blow out our board and you may blow out your power supply! If you are operating a unipolar motor and you short a common lead to a winding pin (WA or WB), then you will blow out our drivers! Similarly, any winding which is shorted to any other winding may burn out our board. If you are setting up to use double-power mode (connecting one motor to both the X and Y motor connectors in order to drive a larger motor), failure to connect the LY- and LY+ limit inputs both to ground will also cause the board to fail. None of these issues are warranted failures; repairs for such are not covered!

After winding lines have been determined, identifying a running sequence can be done by testing the lines using following sequence, connecting to the X motor with clip leads. ***Turn off power*** to the board in between each test, so that power is not on while you change the wiring.

For wires A, B, C, and D (where A, B, C, and D are initially connected to the WA1, WA2, WB1, and WB2 lines) try these orders:

	WA1	***WA2***	***WB1***	***WB2***
1.	A	B	C	D
2.	A	B	D	C
3.	A	D	B	C
4.	A	D	C	B
5.	A	C	D	B
6.	A	C	B	D

For each pattern, request a motor motion in each direction by issuing the command "1000xg0xgi", which should cause the motor to spin to logical location 1000, then back to 0. Wait for the "*" response after each sub-command (the "h", "x", "g", and "i" commands) before typing the next command, in order to let the firmware finish processing the request.

Only when the motor is wired correctly will you get smooth motion first in one direction and then the other.

Once a possible pattern has been determined, you may find that the direction of rotation is reversed from that desired. To reverse the rotation direction, you can either turn the connector around (this may be the easiest method, if a SIP style connector is used), or you can swap both the WA (swap pin 2 with pin 3) and WB pins (swap pin 4 with pin 5). For example, to reverse

A B C D, rewire as

B A D C.

For the purposes of testing, the default power-on rate of 100 ½-steps/second should work with most motors. Otherwise, use the serial connection to define the precise rate needed.

Motor Wiring Examples

Please see the UniversalStepper manual which matches your board hardware for descriptions of how some motors which we tested are wired into the system.

Kit Assembly Instructions

Please see the UniversalStepper manual which matches your board hardware for descriptions of how to assemble your unit.