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

Information and Instruction Manual for BiStepA06 Stepper Motor Controller

By
Peter Norberg Consulting, Inc.
Matches GenStepper Firmware Revisions 1.60-1.70

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 artwork BiStepA06. The firmware releases described are GenStepper versions 1.60 through 1.65. The manual version shown on the front page normally has the same value as the associated GenStepper version. If no manual has yet been published which matches a given firmware level, then the update is purely one of internal details; no new features will have been added.

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

Version	Date	Description
1.59	Sept. 21, 2002	Added SINGLE-MOTOR-DOUBLE-POWER mode
	Oct. 14, 2002	Added docs for BiStepA06 units
1.60	Oct. 20, 2002	Added 'L'atch Report/Reset function
1.63	Dec. 5, 2002	Changed timing of motor winding enables, so 'I'dle wait is immediately valid (no single-step delay), improved motor power at high speeds
1.64	Dec 15, 2002	Code is now fully deterministic on all timings (not just motor winding enables); stepping starts immediately upon receipt of 'G' or 'S'.
1.65	Jan. 21, 2003	Added 'T' command, to allow control of limiT switch detection
1.69	May 14, 2003	Changed all TTL-input sensitivities from Schmitt Triggered to CMOS
1.70	June 9, 2003	Changed 'A'uto full power mode to also disable 'H' mode when the full power rate is reached

The microstep functionality is generated by a PWM (Pulse-Width-Modified)-like algorithm, and is non-feedback based. Although the software has a demonstrated maximum resolution of $1/64^{th}$ of a full-step, in practice most inexpensive stepping motors will not reliably produce unique positions to this level of precision. Mainly, the microstep feature gives you a very smooth monotonic motor action, with the capability of requesting step rates as slow as $1/64^{th}$ of a full step per second. We strongly suggest use of the default $1/16^{th}$ of a full step microstep size; this seems to give the best performance on most motors that we tested. Most non-microstep enabled stepper motors will experience "uneven" step sizes when microstepped between their normal full step locations; however, the steps are monotonic in the correct direction, and are usually consistently located for a given position value.

Product Safety Warnings

The BiStepA06 has 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 (located directly beside the DB9 serial connector)
- The two SN754410 power drivers (both located near the center of the board)
- The PCB board under the SN754410 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

The **BiStepA06** microstepping motor controller from Peter Norberg Consulting, Inc., has the following general performance specifications:

	BiStepA06
Unipolar Motor	Yes
Bipolar Motor	Yes
Maximum Motor supply voltage (Vx and Vy)	34V
Maximum Logic supply voltage (Vc)	15V
Quiescent current (all windings off)	250 mA
Maximum winding current (per motor winding, requires external fan to operate)	1.0A
Board size	2.25" x 3.0"
Dual power supply capable	Yes

Each board can be controlled simultaneously via its TTL input lines and its 2400 or 9600 baud serial interface. If the TTL inputs are used alone, then simple pan, tilt, and rate of motion are provided via 5 input switch closures-to-ground; additional lines are used as limit-of-motion inputs. When operated via the serial interface, full access to the controller's extreme range of stepping rates (1 to 62,500 microsteps per second), slope rates (1 to 62,500 microsteps per second per second), and various motor motion rules are provided. Additionally, a special mode may be enabled which allows an external controller to provide its own step pulses, allowing for up to 62,500 microsteps per second of operation. The boards have a theoretical microstep resolution of 1/64 of a full step, and use a constant-torque algorithm when operating in microstep mode. Please note that, although $1/64^{th}$ resolution is theoretically available, most real use should be restricted to $1/16^{th}$ or $1/8^{th}$ step due to limitations of the non-current feedback PWM stepping methodology used by the code.

The boards themselves have the additional feature of containing provision for in-circuit reprogramming of the Ubicom (Scenix) SX28 chip that is being used as the controller. The Parallax, Inc.^{tm} SX-Key[1] may be used to perform in-circuit reprogramming and debugging of software. ***Note that such action would void the warranty of the product***. This capability is provided as a convenience for those who would like to run different devices (such as three or four phase bipolar steppers) or use different procedures than those for which the product was intended.

[1] Note: SX-Key is a copyrighted product by Parallax, Inc. Please go to their web site at www.parallaxinc.com for more information about this device.

Short Feature Summary

- One or two stepper motors may be independently controlled at one time.
- Each motor may be either Unipolar or Bipolar.
- Each motor may draw up to 1.0 amps/winding. Note that an external cooling fan must be used when your motor draw exceeds 0.4 amps.
- If only a single motor is connected to the board, then you can configure the board to operate in **DOUBLE POWER** mode. This allows the board to operate a single motor at twice the rated current for the board. For example, the BiStepA06 1 amp product can operate a single 2 amp motor, when this feature is enabled.
- Limit switches may optionally be used to automatically request motion stop of either motor 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 $1/64^{th}$ 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 Slew actions are fully supported.
- 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 $1/64^{th}$ 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.
- Simple control of the motors may be done by switch closure. Each motor can be told to slew left or right, or to stop by grounding the relevant input lines. Similarly, the rate of motion can be controlled via stepping through a standard set of rates via grounding another input.
- Complete control of the motors, including total monitoring of current conditions, is available through the 2400 or 9600 baud serial connection.
- An additional mode is available which allows an external computer to directly generate step sequences on the motor control lines. Up to 62,500 steps per second may be requested.
- Runs off of a single user-provided 7.5 to 15 volt DC power supply, or three supplies (7.5-15V for the logic circuits and 7.5-34V for the motors).
- Any number of motors may be run off of one serial line, when used in conjunction with one or more SerRoute controllers.

TTL Mode of operation

The TTL input control method provides for nine input signals and one output signal. TTL based control operates at the same time as serial control; therefore, any of the actions listed below may be requested at any time that the board is not in its special "direct computer control" mode of operation.

All external connections are done via labeled terminal block connections on the left and right hand sides of the boards, and one RS232 serial port on the "bottom" of the board. All of the input and control signals are on the left side, while all of the motor and power connections are on the right side.

TTL Input Voltage Levels: Schmitt-Triggered or CMOS

The input voltage levels which are sensed by the TTL input signals to the boards depend on the firmware version, and the mode of operation of the board.

For all firmware versions from 1.00 through 1.65, all TTL input signals are treated as Schmitt-Triggered levels. This means that voltages <= 0.8 volts generate a logic "0", while voltages >= 4.2 volts generate a logic "1" (assuming that the board power is at 5 volts). Any values in between 0.9 and 4.1 are ignored – the current logic state does not change. This provides extra noise immunity to the system, but turns out to be unneeded except for use by the "1E" state ("Remote Direct Pulse Control"). For safety sake, we suggest a design wherein <=0.4 volts is to be "0", and >=4.6 volts is "1" (for improved noise immunity).

Accordingly, starting with firmware version 1.66 and above, all TTL input signals now are treated as CMOS levels, unless the board is operating in the "1E" state ("Remote Direct Pulse Control"). This means that a logic "0" is generated at any time that the input voltage is <= ½ of the board power, and a logic "1" is generated when the input voltage is above ½ of the board power. Therefore, since our power is 5 volts, a logic "0" is presented when the input is <= 2.5 volts, and a "1" is presented when the signal is above 2.5 volts. In reality, we suggest using <=2 volts for a "0", and >=3 volts for a "1", to avoid any "noise" issues. When the board is in the "1E" state, then it reverts to operating as Schmitt-Triggered (see the prior paragraph), to avoid false-step actions.

Regardless, when doing normal TTL input (such as testing for a "slew X" command), the firmware performs a digital filter on the input signal, to remove possible noise spikes.

Note also that all of the TTL inputs are internally tied to +5 via a very weak resistor (of the order of 5K- to 40K-Ohms). This permits you to use switch-closure-to-board-ground as your method of generating a "0" to the board, with the "1" being generated by opening the circuit.

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. Enabled by default at power on, firmware versions 1.65 and above support the 'T' command, which may be optionally used to enable or disable any combination of these switches.

The connections are:

Signal	*Limit Sensed*
LY-	-Y
LY+	+Y
LX-	-X
LX+	+X

Note that if you wire both the LY- and LY+ lines to ground (so that they are always low when the unit is powered on), then you are telling the firmware to run the X-axis commands to both the X and Y connectors. This allows you to operate a single motor at twice the rated current capacity of the board. Please review the "Single motor, double current mode of operation" section of this manual for more details about this method of operating the board.

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 stop 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. 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.

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, or the "1E" followed by "0E" sequence 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!***

Motor Slew Control: Y- to RDY

Lines Y- through RDY are used to control stepping of the motors, and the rate of steps. The inputs are normally fully debounced (unless running in direct computer control mode), and are designed to operate via a microswitch closure to ground.

The connections are:

Signal	*Action Requested*
Y-	-Y
Y+	+Y
X-	-X
X+	+X
NX	Change Rate
RDY	Motors Ready

When operated normally, the indicated motor is "slewed" in the requested direction at the current rate, as long as the indicated signal is at ground level. Illegal combinations (such as Y- and Y+ both being low at the same time) are treated as "stop", to avoid confusion. As with all other operations of the system, each motor is accelerated to the current rate using the ramp rate defined within the code (which defaults to 4000 microsteps/second/second).

The "Change Rate" action simply selects the "next" rate from its standard internal table of rates, and sets that rate as the requested rate for both motors. The standard rates currently provided after power on reset are:

- 16 microsteps (1 full step)/second
- 40 microsteps (2.5 full steps)/second
- 80 microsteps (5 full steps)/second
- 160 microsteps (10 full steps)/second
- 400 microsteps (25 full steps)/second
- 800 microsteps (50 full steps)/second (this is the power-on default)
- 1600 microsteps (100 full steps)/second
- 4000 microsteps (250 full steps)/second
- 8000 microsteps (500 full steps)/second

Be forewarned that there is no way for the software to tell that a motor cannot operate at a given rate. On power-on, the default microstep is $1/16^{th}$ of a full step; therefore, the default rates range from 1 to 500 full steps/second. Changing the microstep size does change the above real "full step" rates – see the '!' command for more details.

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 when the system is running under "remote pulse control" operation.

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. Any command may be directed to the X, Y or both motors; thus, each motor is fully independently controlled. 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.

Actual control of the stepper motors is performed independently for each motor. A "goto" mode is supported, as is a simple "go in a given direction". The code does support ramping of the stepping rate; however, it does NOT directly support changing the ramp rate, step rate, or "goto" target while a "goto" is under way. The behavior is either that the motor will *first* stop and *then* perform the new request, or that the new parameter value will be used on the *next* action. If button control is performed while a goto is underway, the goto gets changed to a direction slew, and the state of actions is reset.

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

1000G

would mean "go to location 1000"

0G?

would mean "go to location 0, and while that operation was pending, do a diagnostic summary of all current parameters".

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. All firmware versions 1.54 and above handle 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.*** This avoids loss of commands as they are being sent to the control board.

Serial Command Quick Summary

Most of the commands may be preceeded with a number, to set the value for the command. If no value is given, then the last value seen is used.

0-9, +, - – Generate a new VALUE as the parameter for all FOLLOWING commands
A – Select the Auto-Full Power Step Rate
B – Select both motors
E – Enable or Disable Remote Direct Pulse Control
G – Go to position x on the current motor(s)
H – Operate motors at ½ power
I – Wait for motor 'Idle'
K –Set the "Stop oK" rate
L – Latch Report: Report current latches, reset latches to 0
M – Mark location, or go to marked location
O – step mOde – How to update the motor windings
P – sloPe (number of steps/second that rate may change)
R – Set run Rate target speed for selected motor(s)
S – start Slew
T – limiT switch control (firmware versions 1.65 and above)
V – Verbose mode command synchronization
W – Set windings power levels on/off mode for selected motor
X – Select motor X
Y – Select motor Y
Z – Stop current motor
! – RESET – all values cleared, all motors set to "free", redefine microstep. Duplicates Power-On Conditions!
= – Define current position for the current motor to be 'x', stop the motor
? – Report status
other – Ignore, except as "complete value here"

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

Possible combinations:

"-" alone – Set '-' seen, set no value yet: used on SLEW -

"+" alone – Clear '-' seen, set no value yet: used on SLEW+

-n: Value is treated as -n

n: Value is treated as +n

+n: Value is treated as +n

Examples:

-s – Start slew in '-' direction on the current motor

-10s – Slew back 10 steps on the current motor

A – Select the Auto-Full Power Step Rate

This sets the approximate rate (expressed in the current microstep resolution; see the "!" command) at which the system automatically switches to full power to both windings, with strict full-step mode. This is used once the power loss induced by running at high speed becomes significant. *As of firmware version 1.70, this mode will also disable ½ current mode ("1H") once this rate has been reached.*

Note that the code only stores the high byte of this value (i.e., the value divided by 256), and requires that the actual rate divided by 256 be above the value just set. This means that "A" rates of 0-255 all map into 0, and they set all rates 256 and above to be auto-full step mode. **The code defaults at power-on/reset to A=3072** ("3072a"). When the rate is "greater" than 3072, then the motor will run in the full-power, full-step mode. Observe that "A" values of 3072 through 3327 all generate the same test value! When operating at the default microstep resolution of $1/16^{th}$ step size, then the 3072 rate maps into 192 full steps/second. When operating at a microstep resolution of $1/64^{th}$ step size, then the same 3072 rate maps into 48 full steps/second.

For example,

3072A

would set automatic full-power mode to start when the microstep speed exceeds 3072 microsteps/second.

Set this to 62500 to disable this feature.

B – Select both motors

This command selects both the X and Y motors as targets for the following commands.

For example,

B0?

Would generate a report about all reportable parameters for both motors.

At power on/reset, both motors are selected for actions.

E – Enable or Disable Remote Direct Pulse Control

This is used to control whether the TTL input lines are used as direct, edge triggered step requests for their associated motor and direction of travel. The current VALUE is used as the parameter. The options are:

1e – Enable remote pulse control

0e – Disable remote pulse control (the power on/reset default)

When enabled, then all other motor motion commands (such as G and S) have no effect (although changing the step mode, marking locations, and setting rates will affect the stored values for use when remote direct control is disabled). Instead, the TTL input lines are monitored frequently enough to sense 8 microsecond width pulses, looking for low-going-edges (leading edges) in the requests. The leading edges are then used to step the appropriate motor as needed. The stepping actions performed are always in units of the current microstep size, and are masked based on the current winding control rules (see the "!" command for how to control the microstep size, and the "O" command for control of winding/microstepping).

NOTE THAT:

- **THIS COMMAND IS FOR BOTH MOTORS**
- **IT IMMEDIATELY DISABLES ANY PENDING MOTIONS**
- **IF ANY MOTION IS UNDER WAY, THAT FACT IS FORGOTTEN. THIS CAUSES AN INSTANT STOP OF BOTH MOTORS! NO "GRADUAL STOP" (VIA THE AUTOMATIC RAMP MECHANISM) IS PERFORMED. MOTORS OR GEAR TRAINS MAY THUS BE DAMAGED IF THIS IS DONE IMPROPERLY ON SOME SYSTEMS.**
- **TTL INPUTS FOR LIMIT SWITCHES ARE ALSO IGNORED DURING THIS MODE OF OPERATION.**
- **TTL INPUTS ARE TREATED AS "SCHMITT-TRIGGERED" DURING THIS MODE: <=0.4 Volts is "0", >=4.6 volts is "1".**

On both enable and disable, all pending motor actions are **immediately** stopped. The windings on both motors are forced on when remote pulse control is enabled, and are restored to the status defined by the W command when remote pulse control is disabled.

The software switches into a mode of monitoring the TTL inputs very rapidly. It looks for leading (low-going) edges on each of the 4 TTL input lines (low means TRUE, high means FALSE, for compatibility with the normal switch mode of input), and issues a single microstep (in the current microstep precision). The rate of monitoring is such that, if pulses are 8 microseconds wide for each of the high and low states, they will be correctly sensed. Pulse widths less than 8 microseconds may be incorrectly processed! The effective maximum stepping rate is therefore 16 microseconds per microstep (both motors may be stepped at the same time), thus providing for a maximum step rate of 62,500 microsteps per second per motor. Since the maximum microstep rate is ½ full step per microstep, the maximum rate possible with this form of control is 31,250 full steps per second.

G – Go to position x on the current motor(s)

This is used to cause the currently selected motor(s) to travel to the indicated location (from the current Value). The software will:

- Calculate the direction and distance of travel
- Determine how long it has to 'ramp' the motor to go from its current start rate to the standard target rate
- Determine how long it has to then let the motor run at the target stepping rate
- Determine how long it will need to ramp the motor to stop it (which is the same time as that for starting the motor, above).
- Actually perform the action

The code ALWAYS starts from a stop, due to issues of timing. Therefore, if a "Goto" is performed while the motor is running, the system will first stop the motor (as in the 'Z' command), and then restart it based on its then-current location.

For example,

X1000gy-25687g

Would:

1. Select the X motor for actions
2. Start a GOTO on motor X to location 1000
3. Select motor Y for actions
4. Start a GOTO on motor Y to location –25687

Note that the two goto operations continue asynchronously until completed, unless a new command (such as a stop for that motor, or a change in direction request) is received. The current location for a given motor may always be requested, through the "-1" report. For example,

x-1?

Could report

X,-1,350

*

while the motor was still on its way to the requested location.

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.

I – Wait for motor 'Idle'

This allows your code to 'wait' for the currently selected motor(s) to (both) be idle. The code simply waits for either the selected motors to have completed their motion (see the **X**, **Y**, and **B** commands) or for the next serial character to be received, and then it transmits the '*' prompt (ready for next command). Note that, 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 location, and then wait for the motor to actually get there, you could simply issue the command sequence:

Send	Receive
X	*
2000G	* *(note that the '*' is received as soon as the motion starts)*
I	* *(note that this '*' is not received until the motion completes)*

If you send a character before receipt of the final '*' (above), then system will discard transmitting the "*" response if it has not yet started the transmission. It will then process the new character. The best technique to avoid synchronization worries is to send two zero characters ('00'), wait for the second one to be completely sent, and then clear your input buffers. No further characters will be sent from the controller until it sees the next command after this 'flushing' action (i.e., any pending data transmissions will be aborted).

*Please note that if your firmware version is before **1.63**, then you should have one "spacing" character (such as motor selection ('B', 'X', 'Y') or a space) before the 'I', if the immediately prior character was a "S" or "G" (slew or goto). In those versions, it can take up to 1 microstep time for the motor to report that it is "busy". Versions 1.63 and higher mark the motors as busy as soon as the 'S' or 'G' are seen.*

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 max rate (i.e., instant stop) if a value of 0 is requested.

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.

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

For example, after initial power on,

```
L
```

Would report

```
L,16
*
```

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:

```
L,4
*
```

M – Mark location, or go to marked location.

Based on the current parameter value (x), the M command will either cause the selected stepper(s) to record its'/their current position as the "marked" point, or will cause the location to be treated as a "goto" command.

x=0 : Mark current location for a later "go to mark" request

x=1 : Go to last "marked" location

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 16 steps per complete full step, and performs a PWM based near-constant-torque calculation between having 1 or 2 windings enabled at a time. 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 1/64th step near-constant-torque mode (using PWM) – 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.

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 currently have motor X selected, and it is at location 0, then the sequence:

250p500r2000g

would cause the following actual ramp behaviors to occur:

1. The motor would start at its "stop oK" rate, such as 80 microsteps/second

2. It 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).

R – Set run Rate target speed for selected motor(s)

This defines the run-rate to be used for the currently selected motor. It may be specified to be between 1 and 62,500 microsteps per second 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 currently selected motor under the GoTo or Slew 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 $1/64^{th}$ of the specified rate.

For example,

X250RY1000R

Sets the X motor target stepping rate to 250 microsteps per second, and the Y motor target rate to 1000 microsteps per second.

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

S – start Slew.

The "S"lew command is used to cause the currently selected motor to go in the selected direction. If the current value is +/-0 (i.e., just has a sign associated with it), then the motor will slew in the indicated direction on the selected motor(s). Otherwise, the motor(s) will go VALUE steps in the direction indicated by the sign of VALUE, after first stopping the motor (more accurately, will target current location + x, then act as goto).

For example,

+s

will cause the current motor to start slewing in the forward direction.

-250s

will cause the motor to stop, then go 250 steps in the reverse direction.

T – limiT switch control (firmware versions 1.65 and above)

The limi'T' switch command is used to control interpretation of the board limit switch input. **By default (after power on and after any reset action), the board is configured to respond to each of the four limit switches; that is to say, all of the limit switches are enabled.** Control of this feature allows the board to more easily control rotary tables, which may only have an "index" switch instead of a "left and right limit" switch.

Please note that this capability was introduced in firmware version 1.65. It is not available in earlier releases of the firmware.

The command takes a bit-encoded parameter, which lists which switches are to be blocked from action. The values therefore are:

Bit	Numeric Sum Value	Block Limit Detection On…
0	+1	Y-
1	+2	Y+
2	+4	X-
3	+8	X+
Other		RESERVED

For example,

4t

would block detection of the "X-" limit, and allow all of the other limits to work as normal. Similarly,

15t

would block detection of all limits, and

0t

would enable detection of all limits.

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. All firmware versions 1.54 and above handle 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 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. *Firmware versions 1.60 and later also add 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. On firmware versions 1.60 and later, 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, preceeded 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 for selected motor

The "W"indings command controls whether the currently selected motor(s) has its 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 (default setting)

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 – Select motor X

This command selects X motor as the target for the following commands.

For example,

X100r

Would cause the step rate to be set to 100 for motor X.

Y – Select motor Y

This command selects X motor as the target for the following commands.

For example,

Y100r

Would cause the step rate to be set to 100 for motor Y.

Note that if the controller is operating in "**single motor dual power**" mode (selected by grounding both LY- and LY+ during power on/reset operations), then any commands sent to the Y motor controller are effectively ignored. Only the X motor controller sends signals to the X and Y connectors when that mode is enabled.

Z – Stop current motor.

'Z' causes the current motor(s) to be ramped to a complete stop, according to its 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".

For example,

Xz

Would slow down, then stop motor X.

! – 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 32; it is clipped to this range if exceeded.

The suggested values would be the powers of 2, vis. 1, 2, 4, 8, 16, and 32 (giving you true microstep step sizes of 1/64, 1/32, 1/16, 1/8, ¼, and ½ 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 "3/2" in the largest resolution (microstep set to 32).

For example,

4!

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

The reset command also selects the following settings:

- **3072A** – Set the Automatic Full Step rate to be >=3072 microsteps/second
- **B** – Select both motors for the following actions
- **0=** – 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
- **0T** – Enable all limit switch detection
- **1V** – Set <CR><LF> sent at start of new command, no transmission delay time
- **0W** – Full power to motor windings

= – *Define current position for the current motor to be 'x', stop the motor*

This copies the current VALUE as the current position for the selected motors, and then stops said motor(s). For example,

X2000=Y4000=

Would define the current location of the X motor to be 2000, and the current location of the Y motor to be 4000. Note that no actual motor motion is involved – the code simply defines the current location to be that found in the VALUE register, and issues an automatic stop ('Z') request. Note that the motor is stopped AFTER the assignment is complete, so the actual "current position" of the motor will be different from this value, depending on how long it takes for the motor to stop.

X2000=g

Would define the current location of the X motor to be 2000, and then would actually go to that 2000 location. This combination could be used when the motor is actually slewing or executing a "goto", to force the "current" location to be set and selected.

? – 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: ***NOTE THAT ANY LOCATION ABOVE 7 MAY CHANGE BETWEEN CODE VERSIONS***

- 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
- 61: Raw rotor position, Y motor (in 1/64th microstep units)
- 125: Raw rotor position, X motor (in 1/64th microstep units)
- 252: Automatic full-step rate value/256
- 253: Rotor step size, in 1/64th microstep units (see '!' command)

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 location
- -2; Report current speed
- -3; Report current slope
- -4; report target position
- -5; Report target speed
- -6; Report windings state
- -7; report stop windings state
- -8; Report step action (i.e., motor state)
- -9; Report step style
- -10; Report run rate
- -11; Report stop rate
- -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. The letter corresponding to the motor being reported on is sent (i.e., ‘X’ or ‘Y’).
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 is reported.
7. If this is a report for both the X and the Y motors, then a <crlf> is sent.
8. If this is a report for both motors, the other report is sent.
9. If Verbose Mode is on, then a <crlf> is sent
10. A “*” character is sent.

If both motors are being reported, a line containing the X report is sent, followed by a line containing the Y report.

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

Note that in the following examples, first line of “Received” is “*”. This is because two commands are actually being sent (i.e., “B”, then “-<whatever>?”), and each command always generates a “*” response once it has been completed. Technically, fully “synchronized” serial communication consists of (1) send a command, and (2) save all characters until the “*” response is seen. The intervening characters are the results of the command, although only report (“?”) and reset (“!”) generate any significant response.

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 –11. Just after power on, for example, the request of "0?" would generate the report:

X,0,a,b,c,d,e,f,g,h,I,j,k,l,m

Where:

- X is the motor: such as 'X' or 'Y'
- 0 is the report number; 0 is the 'all' report
- a is the value for the current location (report "-1")
- b is the value for the current speed (report "-2")
- c is the value for the current slope (report "-3")
- d is the value for the target position (report "-4")
- e is the value for the target speed (report "-5")
- f is the value for the windings state (report "-6")
- g is the value for the stop windings state (report "-7")
- h is the value for the step action (motor state) (report "-8")
- i is the value for the step style (both full step modes and half) (report "-9")
- j is the run rate (report "-10")
- k is the stop rate (report "-11")

For example,

```
B0?
```

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

```
*
X,0,30,10,1000,30,10,0,0,0,1,100,10
Y,0,-300,10,1000,-300,10,0,0,0,1,100,10
*
```

-1: Report current location

This reports the current (instantaneous) location for the selected motor(s).

For example,

```
B-1?
```

Would report the current location on both motors. You could receive:

```
*
X,-1,10
Y,-1,25443
*
```

-2: Report current speed

This reports the current (instantaneous) speed for the selected motor(s).

For example,

```
B-2?
```

Would report the current speed on both motors. You could receive:

```
*
X,-2,800
Y,-2,2502
*
```

-3: Report current slope

This reports the current (instantaneous) rate of changing the speed for the selected motor(s).

For example,

```
B-3?
```

Would report the current rate on both motors. You could receive:

```
*
X,-3,10
Y,-3,25443
*
```

-4: Report target position

This reports the target location for the selected motor(s).

For example,

```
B-4?
```

Would report the current target on both motors. You could receive:

```
*
X,-4,100
Y,-4,-35443
*
```

-5: Report target speed

This reports the current target run rate which is desired for the selected motor(s). This value is usually either the current stop rate (we are attempting to slow down to this speed) or the current requested run rate (as reported by –10, and as requested by the 'R' command) depending on whether we are speeding up or slowing down.

For example,

```
B-5?
```

Would report the target rate on both motors. You could receive:

```
*
X,-5,800
Y,-5,250
*
```

-6: Report windings state

This reports the current energized or de-energized state for the windings for the selected motor(s). A reported value of 0 means “the windings are off”, a value of 1 means “the windings are energized in some fashion”.

For example,

```
B-6?
```

Would report the current state on both motors. You could receive:

```
*
X,-6,1
Y,-6,0
*
```

-7: Report stop windings state

This reports whether the windings will be left energized when motion completes for selected motor(s). A reported value of 0 means “the windings will be turned off”, a reported value of 1 means “the windings will be left at least partway on”.

For example,

```
B-1?
```

Would report the requested state on both motors. You could receive:

```
*
X,-1,1
Y,-1,0
*
```

-8: Report current step action (i.e., motor state)

This reports the current (instantaneous) state for the selected motor(s). The step action may be one of the following values:

- 0: Idle; all motion complete
- 1: Ramping up to the target speed, in a "GoTo"
- 2: Running at the target speed, in a "GoTo"
- 3: Slowing down, from a "GoTo"
- 4: Slewing ("+s")
- 5: Quick stop in progress ("z", or saw a limit switch closure)
- 6: Reversing direction
- 7: Stopping in preparation for a new GoTo
- 8: Single shot: current action finished (you probably will never see this; it is only selected for about 8 uSeconds)

For example,

```
B-8?
```

Would report the current location on both motors. You could receive:

```
*
X,-8,0
Y,-8,4
```

```
*
```

This would mean that motor X is idle, while motor Y is currently doing some form of slew operation.

-9: Report step style (i.e., micro step, half, full)

This reports the current method of stepping for the selected motor(s). The legal step styles reported are those of the "O" (step mode) command, vis:

- 0: Full step, single windings
- 1: Half step, alternating single/double windings
- 2: Full step, double windings
- 3: Microstep
- +4 added to above: **Single Motor Dual Power** mode is enabled (i.e., both LY- and LY+ were grounded during the last power on/reset of the controller).

For example,

```
B-9?
```

Would report the current stepping method on both motors. You could receive:

```
*
X,-9,3
Y,-9,2
*
```

This would equate to the X motor being in microstep mode, while the Y motor is running in full-power, full step mode.

If you were connected in **dual power mode**, then you could get a report such as:

```
*
X,-9,7
Y,-9,6
*
```

Even though a mode will be reported for the Y motor controller, it is actually ignored in terms of sending signals to the Y motor connector; only the X motor controller affects the signals sent to the X and Y connectors when in dual power mode.

-10: Report run rate

This reports the current requested run rate for the selected motor(s). This is the last value set by the "R" command.

For example,

```
B-10?
```

Would report the current rate on both motors. You could receive:

```
*
X,-10,2000
```

```
Y,-10,3200
*
```

-11: Report stop rate

This reports the speed at which the motors may be considered to be stopped, for starting and stopping activities for the selected motor(s).

For example,

```
B-11?
```

Would report the current stop rate on both motors. You could receive:

```
*
X,-11,80
Y,-11,50
*
```

-12: Report current software version and copyright

This reports the software version and copyright.

For example,

```
B-12?
```

could report:

```
*
genstepper.src $version: 1.48$
Copyright 2002 by Peter Norberg Consulting, Inc. All Rights
Reserved
*
```

other – Ignore, except as "complete value here"

Any illegal command is simply ignored, other than sending a response of "*". However, if a numeric input was under way, that value will be treated as complete. For example,

123 456G

would actually request a "GoTo location 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.

More Examples

For example,

Y 1000 R

Would set the Y rate to 1000 steps/second. The spaces are optional, and would not prevent the code from working; however, an extra "<cr><lf>*" sequence would be sent by the board for each space seen.

B50R

Would set both the Y and X rates to 50 steps per second

300YG

Would go to Y location 300

800G

Would go to location 800 on the most recent motor (in this example, Y)

Y-S

Would start slewing in the minus direction on Y motor

Y+SX3S

Would start slewing positive on Y motor, and would go + 3 steps on the X motor

X1SSS

Would step forward 3 steps on the Y motor, since the calculation is based on the CURRENT TARGET location at the time of the command if the motor is currently executing a GOTO or relative step slew, and is otherwise based on the current MOTOR location. This is thus exactly equivalent to

X3s

X100rY300RB0g

Would cause the step rate to be set to 100 for motor X, 300 for motor Y, and then cause both motors to go to location 0.

Direct TTL Step Control

The '1E' command (see the 'E' command under 'Serial Control') allows a remote controller (another microprocessor, another computer, etc.) to directly request microsteps going in either direction on either (or both) stepper motor(s). The step size used is the current microstep size and is masked based on the current winding control rules (see the "!" command for how to control the microstep size, and the "O" command for control of winding/microstepping). The sampling rate is such that at most 62,500 microsteps/second may be requested on each motor.

NOTE THAT:

- **THIS COMMAND IS FOR BOTH MOTORS**
- **IT IMMEDIATELY DISABLES ANY PENDING MOTIONS**
- **IF ANY MOTION IS UNDER WAY, THAT FACT IS FORGOTTEN. THIS CAUSES AN INSTANT STOP OF BOTH MOTORS! NO "GRADUAL STOP" (VIA THE AUTOMATIC RAMP MECHANISM) IS PERFORMED. MOTORS OR GEAR TRAINS MAY THUS BE DAMAGED IF THIS IS DONE IMPROPERLY ON SOME SYSTEMS.**
- **TTL INPUTS FOR LIMIT SWITCHES ARE ALSO IGNORED DURING THIS MODE OF OPERATION.**

The TTL input lines which are normally used to request a "slew" of a motor in a given direction (when low) get redefined to request a "step" of a motor in a given direction when going low. The wiring thus is:

Signal	*Action Requested*
Y-	-Y microstep
Y+	+Y microstep
X-	-X microstep
X+	+X microstep

The code samples the above lines at a rate such that the minimum time low and minimum time high for each pulse is 8 microseconds (each); shorter pulses may be missed.

A standard sequence to use pulse-based control of the system would thus be:

1. Make certain that the TTL inputs (Y- through X+) are all high.
2. Set up the base microstep size as needed (for example, to step at the maximum precision, issue a "1!" to reset the controller to 1/64 step).
3. Wait about 1/2 second for the reset to complete.
4. Issue the correct winding control command, if needed (by default, the system operates in mode "3o", which is the microstep mode).
5. Issue the "1E" command, to enable TTL based remote control.

6. From now on, until the "0E" (or reset) is issued, a "leading-edge-to-zero" state change on any of the 4 TTL input lines will request a step in the direction of that line.
    - For example, bringing "Y+" low (for at least 5 microseconds) will request a positive (micro) step on the Y motor. The Y+ line must then be brought back high, for at least 5 microseconds, before a new request is guaranteed to be recognized on that line.
    - A motion may be requested at the same time on both the X and Y motors; illegal combinations (such as Y- and Y+ both requesting a step at the same time) are ignored.
    - Note that there is no upper limit on how wide this pulse may be; it just has to be no narrower than 5 microseconds in each direction.

Serial operations which do not request a change in the state of the motor may be processed while running in the TTL mode of control without loss of pulses or steps; however, doing commands which change state may cause lost TTL pulses on inputs and skewing of the PWM signal on outputs.

The following commands will cause up to 16 microseconds of missed TTL control edges during their processing (hence one or two pulses can theoretically be missed). Due to the fact that they are only of use when not in remote TTL control mode, they should not be used in that mode.

- ‘G’ – GoTo
- ‘I’ – wait for motor Idle (during remote TTL control mode, this command never completes)
- ‘M’ – Mark location
- ‘P’ – sloPe rate
- ‘R’ – target Rate
- ‘S’ – start Slew
- ‘Z’ – stop
- ‘W’ – winding mode when stopped (windings are always ON in TTL mode)

The following commands will also cause up to 16 microseconds of missed TTL control edges, and should therefore be used with care. However, they do affect the behavior of the system when in remote TTL control mode, and hence may be of use.

- ‘H’ – Half power
- ‘O’ – step mOde
- ‘=’ – set location
- ‘!’ – Reset the controller; abort all actions, restart system.

All of the other commands may be used with no negative effects on timing in the system.

Basic Stamp™ Sample Code

The UniversalStepper series of boards may all be used with the Parallax, Inc.tm Basic Stamp tm series of boards. The connection to the UniversalStepper product is usually via the terminal block connector which contains the RDY, SI, and SO signals, with the JS jumper removed. The remaining input pins on the input set of connectors may be wired or not, as needed by the application. Most of the time, they will be left unconnected (to "float").

Communications between the two boards may be performed either at 9600 baud (the default), or 2400 baud (via a configuration option). Normally, operating at the 9600 baud rate is recommended; use the 2400 baud rate only if you cannot make your code work at 9600 baud. You **must** use the 'V'erbose command to configure the controller to pause one character time before sending responses to the Basic Stamp, to avoid data synchronization issues. If 2400 baud communications is desired, connect a ¼ (or 1/8) watt 1K resistor between the RDY (B5) signal and ground (GND). The UniversalStepper samples this line as an input during reset or restart, and treats a "low" as a request for 2400 baud operation.

The sample code provided by Peter Norberg Consulting, Inc. assumes that the following connections have been made between the UniversalStepper and the Basic Stamp:

- RDY (B5) connected to P3
- SI (Serial Output, B6) connected to P2
- SO (Serial Input, B7) connected to P1

Some of the code provided operates at 2400 baud. To do so, a 1K resistor is also wired between RDY/P3 and GND, which thus provides the required "low" signal to READY during the reset operation. Note that, in reality, all of the code can run correctly at 9600 baud on most stamps; operation at 2400 baud is shown here just to demonstrate the technique.

"Gendemo.bs2" is a 9600 baud demo, which uses the RDY line for synchronization. It runs using a microstep size of 4/64 (1/16) of a full-step, and constantly spins both motors between logical position 2000 and 0. On each "spin cycle", the stepping mode gets changed; each of the legal stepping modes (full step 2-winding, full step 1-winding, ½ step, and microstep) are exercised in sequence, and a 1/5 of a second pause is inserted between each cycle for ease of visual synchronization.

"Gendemoser.bs2" is a 2400 baud demo, which ignores the "RDY" line and uses the SERIAL input line for all of its synchronization. Aside from operating strictly using the serial communications interface, it operates identically to "Gendemo.bs2". Since it operates at 2400 baud, it requires the 1K resistor-to-GND on the P3/RDY line.

"Genseekser.bs2" is a somewhat more comprehensive example, in terms of showing the capabilities of the UniversalStepper system. As with "Gendemoser.bs2", this operates at 2400 baud (thus needing the 1K resistor). It operates at the full level of microstep possible (1/64 of a full step), and runs each motor at a different speed. X is set to a maximum rate of 4000 microsteps/second (which is 4000/64 or 62.5 full steps/second), with a matching ramp rate of 4000 microsteps/second/second. Y is set to a maximum rate of 8000 microsteps/second (which is 125 full steps/second), with a ramp rate of 7000

microsteps/second/second. It also sets the "automatic full-power" step rate to be 6000 microsteps/second. Given that only Y will exceed this rate, the Y motor will switch from what ever mode it is using to full power mode during any seek which goes far enough for it to exceed the 6000 microsteps/second rate. Having gone through this setup, the loop operates similarly to that in "Gendemoser.bs2", except that the locations cycled are +16,000 and 0. If you use this demo with two identical motors, you should be able to "hear" the difference in the stepping modes, and you should also hear the Y motor "become noisy" partway through the microstep phase of the entire sequence (when it switches between microstep mode and full power full step mode).

The complete sources to these examples are installed by default into the "C:\UniversalStepper" directory, when you install the code provided with the product.

Listing for GENDEMO.BS2 – 9600 Baud, READY line based

```
' ****************************************************************************
' $modname: gendemo.bs2$
' $nokeywords$
' Demonstrates some of the serial commands using goto and TTL Busy line to the
' SimStep and BiStep
' set of controllers from Peter Norberg Consulting, Inc.
'
' The tool first initializes the stepper to operate at 16 microsteps/full step,
' with the start/stop rate being 80 uSteps/second, and the ramp rate at
' 1000 uSteps/sec/sec.
' The target ramp rate is 1000 uSteps/second;
' The auto-power switch mode (the 'A' command) is left at its default of 3072,
' which is equivalent to 192 full
' steps/second.
'
' Note that both motors are selected for the actions by default.
' It then enters the speed test loop.
'
' The code first waits for the stepper unit to report idle.
' and it is instructed to move to logical location 2000 (in) 1/16th steps.
' (Note that this is full step location 62.5).
' This is then followed by a move to location 0, and then a new stepping mode
' is selected.  A 1/5th second pause is inserted to make it easy to identify
' when the cycle is occurring.  All three modes of stepping are cycled:
'   Mode    Use
'    0      Single Winding mode (1/2 power full steps)
'    1      Half step mode (alternate single/double windings on)
'    2      Full step mode (double windings on)
'    3      Microstep mode (full microstep processing; DEFAULT MODE)
'
'
' SPECIAL TIMING NOTE: It can take the SimStep/BiStep up to 100 uSeconds to respond to
' a new serial "go" command (goto or slew); therefore, you must always wait
' a small amount of time (at least a few milliseconds uSecs) before testing the
' "busy" line, since
' you may get a "false idle" response.
'
' Additional note: The SimStep/BiStep products operate at 9600 baud.  Although
' the Basic Stamp series can send this rate reliably,  many of them cannot receive
' at this rate without data loss; therefore, no attempt is made in this
' sample to receive serial data from the controller.
' ****************************************************************************

' {$STAMP BS2}


' SimStep or BiStep connected as follows
'   Serial Input P1 to SimStep B7 Serial output
'   Serial Output p2 to SimStep B6 Serial Input
'   busy          p3 to SimStep B5 Status Output
'                 (HIGH = idle, LOW = motion in progress)
'                 AND busy NOT connected to 1K resistor to ground (force 9600 baud)

PortStepperSerFrom  con  1      ' Serial from stepper port
PortStepperSerTo    con  2      ' Serial to stepper port
PortStepperBusy     con  3      ' Busy line
PortStepperBaud     con 84      ' Baud rate to generate 9600 baud:
                                ' Must have no pull-down resistor on busy line!

PortStepperBusyTest var in3    ' Same as PortStepperBusy, used for input test

idMicroStep var byte            ' Gets microstep mode; cycles 0 to 3


' Code restarts here if RESET button pressed

        input             PortStepperBusy         ' BUSY from stepper

        pause 250                                  ' Wait for stepper power on cycle
        serout PortStepperSerTo,PortStepperBaud,["4!"]
                          ' Reset the stepper, set 4/64 full-step step size
```

```
        pause 1000
                  ' Wait for stepper to send its wake-up copyright text
        serout PortStepperSerTo,PortStepperBaud,["80K"]
                  ' Set Stop OK to 'can start/stop at 80 microsteps/sec'
        serout PortStepperSerTo,PortStepperBaud,["1000p"]
                  ' For demo purposes, set a slow ramp of 1000 microsteps/sec
        serout PortStepperSerTo,PortStepperBaud,["1000R"]
                  ' For demo purposes, set a target rate of 1000 microsteps/sec
        idMicroStep = 0
                  ' Start at microstep 0


loop:
        serout PortStepperSerTo,PortStepperBaud,[dec idMicroStep,"o"]
                  ' Set microstep mode

        serout PortStepperSerTo,PortStepperBaud,["2000g"]
                  ' Go to location 2000

        gosub WaitReady
                  ' Wait until ready

        serout PortStepperSerTo,PortStepperBaud,["0g"]
                  ' Go back to 0

        idMicroStep = (idMicroStep + 1) & 3
                  ' Cycle step type
        gosub WaitReady                    ' Wait until ready
        pause 200                          ' wait 0.2 seconds before we cycle
        goto        loop                   ' Cycle forever

WaitReady:
        pause 100    ' Wait 0.1 seconds for prior character to be processed
        if PortStepperBusyTest = 0 then WaitReady       'Wait till not busy
        return
```

Listing for GENDEMOSER.BS2 – 2400 baud, serial based

```
' ****************************************************************************
' $modname: gendemoser.bs2$
' $nokeywords$
' Demonstrates some of the serial commands using goto and serial response to
' the SimStep and BiStep
' set of controllers from Peter Norberg Consulting, Inc.
'
' The tool first initializes the stepper to operate at 16 microsteps/full step,
' with the start/stop rate being 80 uSteps/second, and the ramp rate at
' 1000 uSteps/sec/sec.
' The target ramp rate is 1000 uSteps/second;
' The auto-power switch mode (the 'A' command) is left at its default of 3072,
' which is equivalent to 192 full
' steps/second.
'
' Note that both motors are selected for the actions by default.
' It then enters the speed test loop.
'
' The code first waits for the stepper unit to report idle.
' and it is instructed to move to logical location 2000 (in) 1/16th steps.
' (Note that this is full step location 62.5).
' This is then followed by a move to location 0, and then a new stepping mode
' is selected.  A 1/5th second pause is inserted to make it easy to identify
' when the cycle is occurring.  All three modes of stepping are cycled:
'   Mode    Use
'    0      Single Winding mode (1/2 power full steps)
'    1      Half step mode (alternate single/double windings on)
'    2      Full step mode (double windings on)
'    3      Microstep mode (full microstep processing; DEFAULT MODE)
'
'
' SPECIAL TIMING NOTE: It can take the SimStep/BiStep up to 100 uSeconds to respond to
' a new serial "go" command (goto or slew); therefore, you must always wait
' a small amount of time (at least a few milliseconds uSecs) before testing the "busy"
' line, since
' you may get a "false idle" response.
'
' Additional note: The SimStep/BiStep products normally operate at 9600 baud.
' Although
' the Basic Stamp series can send this rate reliably,  many of them cannot receive
' at this rate without data loss; therefore, a special patch has been made available
' to the GenStepper versions 2.41 and later, to allow optional selection for a 2400
' baud rate.  This patch is to connect a 1K resistor from the busy line (SimStep B5
' status output)
' to ground -- the GenStepper source detects this pull-down during its initialization
' sequence, and then runs at 2400 baud.
' ****************************************************************************

' {$STAMP BS2}


' SimStep or BiStep connected as follows
'   Serial Input P1 to SimStep B7 Serial output
'   Serial Output p2 to SimStep B6 Serial Input
'   busy          p3 to SimStep B5 Status Output
'                    (HIGH = idle, LOW = motion in progress)
'                    AND busy connected to 1K resistor to ground (force 2400 baud)

PortStepperSerFrom  con  1      ' Serial from stepper port
PortStepperSerTo    con  2      ' Serial to stepper port
PortStepperBusy     con  3      ' Busy line
PortStepperBaud     con 396     ' Baud rate to generate 2400 baud:
                                ' Must have pull-down resistor on busy line!

PortStepperBusyTest var in3    ' Same as PortStepperBusy, used for input test

idMicroStep var byte            ' Gets microstep mode; cycles 0 to 3
'szSerString var byte(2)        ' Only used if you enable debug mode (see comments)


' Code restarts here if RESET button pressed
```

```
        input             PortStepperBusy               ' BUSY from stepper

        pause 250
           ' Wait for stepper power on cycle
        serout PortStepperSerTo,PortStepperBaud,["4!"]
           ' Reset the stepper, set 4/64 full-step step size
        pause 1000
           ' Wait for stepper to send its wake-up copyright text
       serout PortStepperSerTo,PortStepperBaud,["2V"]
           ' Set short responses, but add delay before response
        serout PortStepperSerTo,PortStepperBaud,["80K"]
           ' Set Stop OK to 'can start/stop at 80 microsteps/sec'
        serout PortStepperSerTo,PortStepperBaud,["1000p"]
           ' For demo purposes, set a slow ramp of 1000 microsteps/sec
        serout PortStepperSerTo,PortStepperBaud,["1000R"]
           ' For demo purposes, set a target rate of 1000 microsteps/sec
        idMicroStep = 0
           ' Start at microstep 0


loop:
        serout PortStepperSerTo,PortStepperBaud,[dec idMicroStep,"o"]
           ' Set microstep mode

        serout PortStepperSerTo,PortStepperBaud,["2000g"]
           ' Go to location 2000

        gosub WaitReady                                    ' Wait until ready

        serout PortStepperSerTo,PortStepperBaud,["0g"]     ' Go back to 0

        idMicroStep = (idMicroStep + 1) & 3 ' Cycle step type
        gosub WaitReady                                    ' Wait until ready
        pause 200                          ' wait 0.2 seconds before we cycle
        goto        loop                   ' Cycle forever

WaitReady:
'        DEBUG "Waiting..."
        serout PortStepperSerTo,PortStepperBaud,["00I"]
           ' wait for ready; the leading 0's flush BiStep's output queue

       SerIn PortStepperSerFrom,PortStepperBaud,[WAIT("*")]    ' And wait for done
'        SerIn PortStepperSerFrom,PortStepperBaud,[STR szSerString\1]
'     DEBUG "Saw: ", STR szSerString, "[", HEX szSerString(0), "]", CR
        return
```

Listing for GENSEEKSER.BS2 – 2400 Baud, serial based, complex actions

```
' ************************************************************************
' $modname: genseekser.bs2$
' $nokeywords$
' Demonstrates some of the serial commands using seek and serial response
' to the SimStep and BiStep
' set of controllers from Peter Norberg Consulting, Inc.
'
' The tool first initializes the stepper to operate as follows:
'   64 microsteps/full step,
'   start/stop rate being 320 uSteps/second
'   ramp rate at 4000 uSteps/sec/sec for the X motor, 7000 uSteps/second for the
'   Y motor.
'   Auto-power switch mode (the 'A' command) is reset to 6000 uSteps/second
'   Target ramp rate is 4000 uSteps/second for X, 8000 uSteps/second for Y
'
' This combination means that the X motor will peak at 1/2 the speed of the Y motor,
' and that the Y motor will switch to full-step full power mode during the midpoint
' of the seek.
' During the microstep pass test (when idMicroStep = 3), you will notice that the
' Y motor
' will start quietly, and then suddenly become noisy for a short period, and then
' it will quiet
' down again.  This is occurring when the stepping mode switches from micro to
' full when
' the motor speed is faster than about 6000 uSteps per second.
'
'
' Note that both motors are selected for the seek actions.
' It then enters the speed test loop.
'
' The code first waits for the stepper unit to report idle.
' and it is instructed to move +16000 (in) 1/64th steps.
' (Note that this is full step delta 125).
' This is then followed by a move to location -16000, and then a new stepping mode
' is selected.  A 1/5th second pause is inserted to make it easy to identify
' when the cycle is occurring.  All three modes of stepping are cycled:
'   Mode    Use
'    0      Single Winding mode (1/2 power full steps)
'    1      Half step mode (alternate single/double windings on)
'    2      Full step mode (double windings on)
'    3      Microstep mode (full microstep processing; DEFAULT MODE)
'
'
' SPECIAL TIMING NOTE: It can take the SimStep/BiStep up to 100 uSeconds to respond to
' a new serial "go" command (goto or slew); therefore, you must always wait
' a small amount of time (at least a few milliseconds uSecs) before testing the
' "busy" line, since
' you may get a "false idle" response.
'
' Additional note: The SimStep/BiStep products normally operate at 9600 baud.
' Although
' the Basic Stamp series can send this rate reliably,  many of them cannot receive
' at this rate without data loss; therefore, a special patch has been made available
' to the GenStepper versions 2.41 and later, to allow optional selection for a 2400
' baud rate.  This patch is to connect a 1K resistor from the busy line
'  (SimStep B5 status output)
' to ground -- the GenStepper source detects this pull-down during its initialization
' sequence, and then runs at 2400 baud.
'
' Since this is a relative seek on both motors, you can test the limit switches
' easily;
' just ground one of the limit inputs (A0-A3) at a time, and observe which motor stops
' going
' which direction.
'
'       Ground  Direction
'       Line    Blocked
'       A0      -Y
'       A1      +Y
'       A2      -X
'       A3      +X
```

```
'
' ****************************************************************************

' {$STAMP BS2}


' SimStep or BiStep connected as follows
'   Serial Input P1 to SimStep B7 Serial output
'   Serial Output p2 to SimStep B6 Serial Input
'   busy          p3 to SimStep B5 Status Output
'                  (HIGH = idle, LOW = motion in progress)
'                  AND busy connected to 1K resistor to ground (force 2400 baud)

PortStepperSerFrom  con  1      ' Serial from stepper port
PortStepperSerTo    con  2      ' Serial to stepper port
PortStepperBusy     con  3      ' Busy line
PortStepperBaud     con 396     ' Baud rate to generate 2400 baud:
                                ' Must have pull-down resistor on busy line!

PortStepperBusyTest var in3    ' Same as PortStepperBusy, used for input test

idMicroStep var byte            ' Gets microstep mode; cycles 0 to 3
'szSerString var byte(2)        ' Only used if you enable debug mode (see comments)


' Code restarts here if RESET button pressed

        input             PortStepperBusy               ' BUSY from stepper
        pause 250
           ' Wait for stepper power on cycle
        serout PortStepperSerTo,PortStepperBaud,["1!"]
           ' Reset the stepper, set 1/64 full-step step size
        pause 1000
           ' Wait for stepper to send its wake-up copyright text
        serout PortStepperSerTo,PortStepperBaud,["2V"]
           ' Set short responses, but add delay before response
       serout PortStepperSerTo,PortStepperBaud,["320K"]
           ' Set Stop OK to 'can start/stop at 320 microsteps/sec'
        serout PortStepperSerTo,PortStepperBaud,["6000A"]
           ' Set auto-switch to full power mode to 6000 microsteps/sec;
           ' only Y will do it
        serout PortStepperSerTo,PortStepperBaud,["X"]
           ' For demo purposes, Select just X for a moment
        serout PortStepperSerTo,PortStepperBaud,["4000p"]
           ' For demo purposes, set X slow ramp of 4000 microsteps/sec
        serout PortStepperSerTo,PortStepperBaud,["4000R"]
           ' For demo purposes, set X target rate of 4000 microsteps/sec
        serout PortStepperSerTo,PortStepperBaud,["Y"]
           ' For demo purposes, Select just Y for a moment
        serout PortStepperSerTo,PortStepperBaud,["7000p"]
           ' For demo purposes, set Y faster ramp of 7000 microsteps/sec
        serout PortStepperSerTo,PortStepperBaud,["8000R"]
           ' For demo purposes, set Y target rate of 8000 microsteps/sec
        serout PortStepperSerTo,PortStepperBaud,["B"]
           ' For demo purposes, Select both X and Y for remaining actions
        idMicroStep = 0
           ' Start at microstep 0


loop:
        serout PortStepperSerTo,PortStepperBaud,[dec idMicroStep,"o"]
           ' Set microstep mode

        serout PortStepperSerTo,PortStepperBaud,["+16000s"]
           ' Go forward 16000 (real "full step" loc = 16000/64 = 250)

        gosub WaitReady                                    ' Wait until ready

        serout PortStepperSerTo,PortStepperBaud,["-16000s"]
           ' Go back to 0

        idMicroStep = (idMicroStep + 1) & 3 ' Cycle step type
        gosub WaitReady                                    ' Wait until ready
        pause 200                          ' wait 0.2 seconds before we cycle
```

```
        goto        loop                    ' Cycle forever

WaitReady:
'        DEBUG "Waiting..."
        serout PortStepperSerTo,PortStepperBaud,["00I"]
           ' wait for ready; the leading 0's flush BiStep's output queue

       SerIn PortStepperSerFrom,PortStepperBaud,[WAIT("*")]    ' And wait for done
'        SerIn PortStepperSerFrom,PortStepperBaud,[STR szSerString\1]
'     DEBUG "Saw: ", STR szSerString, "[", HEX szSerString(0), "]", CR
        return
```

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 UniversalStepper 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.

- /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 UniversalStepper 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 4!x1000gy-2000gi

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 /pCOM2 /b2400 /i10000 y+5000s

Would:

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

BiStepComCtl.dll – A COM controller for UniversalStepper

The BiStepComCtl.dll object is a sample Visual Basic COM application which allows any COM-aware system (such as VBScript based scripts) to easily control the UniversalStepper 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 its main source (bistepcom\bistepcom.cls); see that source for details of all of its features. From the point of view of using the source, however, the code provides access to most of its features via use of the SAME interface as that provided by the SerTest application. That is to say, it interprets text sent to it by applications to see if they are command switches; if they are not, then they get sent on to the currently selected serial port. If they are, then they are interpreted in the same manor as the SerTest tool.

The most important functions provided by the BiStepComCtl control are:

- Public Function StringWriteCommandSequence(ByRef strBuffer As String) As Long

  This is the primary method for sending a complete set of commands to the controller. The code first clears all internal response buffers (the same action as 'AllResponseClear', 'ActionLogClear', and 'ErrorLogClear', below) and then sends each command in the string as needed. It fully parses the return values into an internal member structure, which is available via the "ResultGet" function.

  The strBuffer sent may either be a series of commands to send to the controller, or one of the "switch" based commands (/bBaudRate, /pSerialPort, or /iIdleTime, as described in the SerTest section)

- Public Function ResultGet(ByRef strWhichMotor As String, ByRef strItem As String)

  This is the primary method used to get responses back from the remote controller. BiStepCom maintains the last value seen for each of the legal "report" commands (see the "? – Report Status" section of this manual) for each motor. The strWhichMotor selects the motor to access ("X" or "Y"), while the strItem is the item identifier for the report ("-1" would therefore be the selected motor's current location).

- Public Property Get ActionLog() As String

  This reports the COMPLETE list of commands sent and responses received, since the control was loaded, or since the last call to "ActionLogClear".

- Public Sub ActionLogClear()

  This clears the action log

- Public Property Get AllResponse() As String

  This extracts the complete set of responses to the most recent "StringWriteCommandSequence" sent (or, more accurately, the complete

set of responses to the internal StringWrite function calls which are generated by the last StringWriteCommandSequence seen).

- Public Sub AllResponseClear()

  This clears the AllResponse buffer

- Public Property Get Copyright() As String

  This reports the most recent "Copyright" response from the controller.

- Public Property Get ErrorLog() As String

  This extracts a complete list of all error logging information accumulated so far.

- Public Sub ErrorLogClear()

  This clears the error logging information

- Public Get LastCommaResult()

  This retrieves the last primary result of a comma-separated response. For example, if the controller has sent a "R,-3,6", this would report the "-3". Similarly, if the controller has sent the latch-response "L,16", this would report the "16" value.

- Public Function FlushInput As Long

  This clears the serial input buffer, and should be used when starting up the control to make certain that there are no "left-over" characters in the input buffer, which could confuse the software's synchronization.

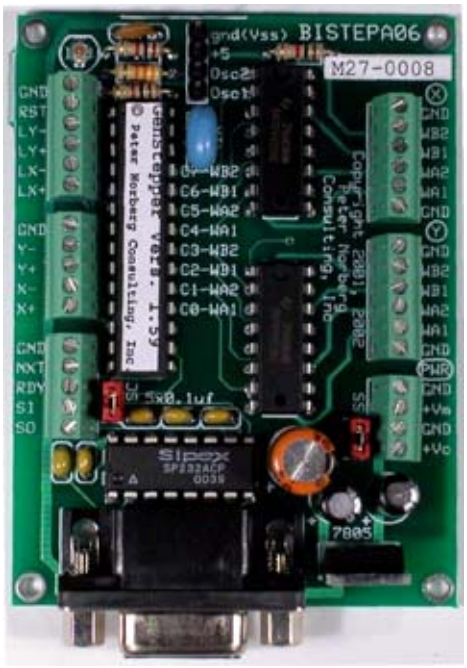
Bistepclass.vbs – A sample script using BiStepComCtl

The "bistepclass.vbs" script is a simple, complete example which shows use of the BiStepComCtl.dll object. Its' listing follows:

```
'************************************************************
' Function to test BiStepComCtl object
'************************************************************

Option Explicit


    Dim strResponse
    Dim strSep
    Dim hBiStep                                                 ' Our local hBiStep

    Set hBiStep = CreateObject("BiStepComCtl.BiStepCom")        ' Create our object

    strSep = vbCrLf & String(20, "*") & vbCrLf

    hBiStep.FlushInput                  ' Attempt to flush the input
    hBiStep.StringWriteCommandSequence ("/i6000")       ' Set the idle time to 6 secs
    hBiStep.StringWriteCommandSequence ("4!X1000GY-3000GBI0?")
           ' Go to location X loc 1000*4/64, Y loc -3000*4/64,
           ' and wait for idle on both motors, then query status
    strResponse = "Current Action log is" & strSep & hBiStep.ActionLog & strSep
    strResponse = strResponse & "Current Error Log is " & strSep _
           & hBiStep.ErrorLog & strSep
    strResponse = strResponse & "Last Response was " & strSep _
           & hBiStep.LastResponse & strSep
    strResponse = strResponse & "All response was " & strSep _
           & hBiStep.AllResponse & strSep
    WScript.StdOut.Write strResponse
    WScript.Echo hBiStep.Copyright
    WScript.Echo "Type,X,Y"
    Dim idWhich
    For idWhich = -1 to -11 Step -1
           WScript.Echo idWhich & "," _
                  & hBiStep.ResultGet("X", idWhich) & "," _
                  & hBiStep.ResultGet("Y", idWhich)
    Next
    hBiStep.ActionLogClear
    hBiStep.ErrorLogClear
    ' Now, just to show how it works, let us extract the rotor step size,
    ' and the X and Y raw rotor positions

    hBiStep.StringWriteCommandSequence ("X253?125?Y61?")

    ' Then we report the values from the above query

    WScript.Echo "Rotor step size = " & hBiStep.ResultGet("X", 253)
    WScript.Echo "  X raw rotor position = " & hBiStep.ResultGet("X", 125)
    WScript.Echo "  Y raw rotor position = " & hBiStep.ResultGet("Y", 61)

    hBiStep.ActionLogClear
    hBiStep.ErrorLogClear
    Set hBiStep = Nothing
```

Board Connections

The current BiStepA06 board is as shown as follows.



Board Size

The board, oriented as shown on this page, is 3.0 inches high by 2.25 inches wide.

Mounting Requirements

The board may be mounted using four 2-56 or 2-64 machine screws. The holes are positioned exactly 0.1 inches in from each corner. Thus, vertically their centers are 2.8 inches apart, and horizontally they are 2.05 inches apart.

Connector Signal Pinouts

There are eight connectors on each board.

Going from top-left down, we have:

- SX-Key debugger connector (4 pin SIP header)
- SX-28 Direct Access signals:
  - TTL Limit Input and RESET (GND, RST, LY- to LX+)
  - TTL Motor Direction Slew Control (Y- to X+)
  - Board status and TTL Serial (NX, RDY, SI (serial input), SO (serial output)). Only use the TTL serial if the JS jumper, located near the bottom-left portion of the board, is removed.

Then, on the bottom we have:

- RS232 Serial, as a DB9 female connector on the bottom of each board.

Finally, going from top-right down, we have:

- X Motor connector (upper right)
- Y Motor connector (center)
- Power connector (lower right: provides separate motor and logic power)

SX-Key debugger connector

Pin	Name	Description
1	GND	Vss (gnd) for SX-Key
2	+5V	+5 for SX-Key
3	OSC2	Oscillator Input 2
4	OSC1	Oscillator Input 1

This connector allows use of the Parallax, Inc.[tm] SX-Key debugger/programmer product, to reprogram the SX-28 in place. ***If the SX-Key is used as a debugging device, then the resonator (XTL) MUST BE REMOVED, or damage to the SX-Key may occur!***

TTL Limit Input and Reset

Name	Description
GND	Ground reference for inputs – short input to GND to denote limit
RST	Reset the microcontroller, when low
LY-	Y Minimum limit reached, when low
LY+	Y Maximum limit reached, when low
LX-	X Minimum limit reached, when low
LX+	X Maximum limit reached, when low

This portion (the next 5 pins) of the J1 connector is used to warn the SX-28 that a limit is being reached in the given direction. The signals are designed to be shorted to GND to denote that a limit is present; they are also compatible with normal TTL outputs from any TTL compatible device. LY- through LX+ are internally pulled up to +5 with 10K resistors (within the SX-28 itself).

TTL Motor Direction Slew Control

Name	Description
GND	Signal ground
Y-	Slew Y Negative
Y+	Slew Y Positive
X-	Slew X Negative
X+	Slew X Positive

This connector gives access to the TTL motor direction control signals for the system.

Y- through X+ are inputs, used to control manual slew requests. They each cause the indicated motor to turn at its current rate in the indicated direction, as long as the indicated signal is grounded. For example, connecting pin Y- to GND (or providing a low TTL input signal) will cause the Y motor to go in the “negative” direction.

Board status and TTL Serial

Name	Description
GND	Ground reference for all signals
NXT	Go to "next" step rate
RDY	Ready/busy output
SI	INPUT: Raw SX-28 Serial Input (TTL level)
SO	OUTPUT: Raw SX-28 Serial Output (TTL Level)

This connector gives access to the serial control signals for the SX-28, as well as board status and slew rates.

NXT is used to select the "next" rate, stepping through a standard list of rates each time the input is grounded (or is driven low as a TTL input).

RDY is normally an informational output that describes the state of "one or more motors are still stepping". High means READY/IDLE, low means STEPPING. ***During processor reset, RDY is sampled as an input – if it has been tied to ground via a 1K resistor, then the code selects a communication baud rate of 2400 baud. Otherwise, 9600 baud is selected.***

SI and SO are the "real" serial input and serial output (respectively), as seen by the SX-28 chip. If the application desires direct serial communications without RS232 levels (for example, if the Parallax Inc.[tm] Basic Stamp [tm] based products are being used to control the board), simply remove the JS jumper (the jumper near the connector containing the SI/SO signals) and use these pins.

Note that if this board is a "child" board in a SerRoute controlled tree of boards, then the SS jumper will normally be removed, unless special interfacing is done.

The communication rate is fixed at 2400 or 9600 baud, no parity, 8 data bits, 1 stop bit. As noted above, RDY is used as an input during reset to determine the baud rate – if tied to ground via a 1K resister, then the baud rate is set to 2400 baud. If left to float (the default), then the baud rate is set to 9600 baud.

RS232 Serial DB9 Female (socket)

Name	Description
SO	Serial Output – To External Computer Serial Input
SI	Serial Input – From External Computer Serial Output
GND	Signal Ground

This connector provides for all external serial communications, using the RS232-C standard. It is directly compatible with a normal male/female DB9 connection to a computer.

Power Connector (labeled here top-to-bottom) And Motor Voltages

Name	Description
GND	Ground for Vm
+Vm	4.5-34 volts, for the X and Y motors
GND	Ground for Vc
+Vc	+7.5-15 volts for the logic circuits

The power connector has two sets of power and ground pins. This is mainly to make it possible to deliver power to high-voltage motors (i.e., any motor which needs more than 15 volts) while still powering the logic circuit off of its required 7.5 to 15 volts.

There are several ways of powering the system, which are dependent upon the current and voltage requirements of the system. One or two power supplies may be used, depending upon the voltage needed by the motors and upon whether extra cooling can be applied to the 7805 voltage regulator. If the motors require more than 15 volts to operate, **using the same power supply to the 7805 will cause the 7805 to get extremely hot** (over 100 deg. C). Although it technically can withstand temperatures up to 150 deg. C, we do not recommend or warrant it. It is much better in this case to split the supplies. Use pins 3 (GND) and 4 (+Vc) to provide 7.5 to 15 volts to the 7805 (the lower voltage you use, the better it is from a heat point of view) at 150 or 250 mA (150 for the SimStep, 250 for the BiStep). Use pins 1 (GND) and 2 (+Vm) to provide the motor power in this case, and ***REMOVE THE "SS" JUMPER WHICH IS NEAR THE POWER CONNECTOR—OTHERWISE YOU WILL BE SHORTING THE POWER SUPPLIES TOGETHER***!

5 volt motors may be operated, although the exact voltage being provided to the motor may be somewhat uncertain. If you use a single power supply, the supply must be 7.5 volts (so that 5 volts will be provided to the logic circuits). In this case, the motor will be supplied with 5.5-6.4 volts, depending on temperature and particular parts. If you split the supplies, then the motor supply (pins 2-3) can be "tweaked" to determine the best voltage for your motor – it will be in the range of 6.1 to 7 volts, assuming that you do not want to exceed the 5 volt specification for the motor.

The power options can thus be summarized as:

Motor Voltage	SS Installed	Use sep. Power supply	Comments
6-15V	YES	NO	Single power supply, connected to pins 1 (GND) and 2 (Vm) of power connector. SS Jumper is installed.
<6 or >15V	NO	YES	Use two power supplies, one for the motor (connected to pins 1 (GND) and 2 (Vm)), the other for the digital power (pins 3 (GND) and 4 (Vc)). The digital power should be 7.5 to 15 volts, at least 300 ma. The SS jumper is removed.

Note also that if the current requirements are over about 0.4 Amp/winding, then fan-based cooling of the board is usually required. The driver components can get quite hot, and external cooling will increase their lifetime considerably!

Fan based cooling should be done such that the bottom and top of the board in the area of the SN754410 components are both exposed to about 8-10 CFM of air flow. A single side-positioned fan, which directs air over both sides of the board (top and bottom) is usually the easiest way to achieve this type of flow.

Calculating Current 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	2.0

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.

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.

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 of the other units, 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.

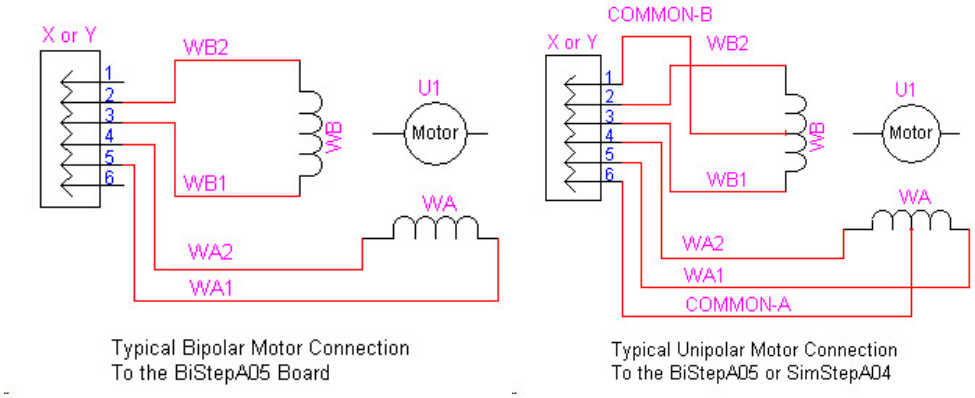
The motor supply should be above 7 volts in all cases (due to some signal requirements on the board), and otherwise is as calculated under sections 1 through 3, above. If the supply is to drive 2 motors, please remember to double the current needs.

Tripple 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. Note, however, that if you operate the BiStep2A in DOUBLE POWER mode, then one supply must be used to operate the power for both motor driving circuits: otherwise, you will cause failure of the board and/or the power supplies!

X and Y Motor Connectors, Wiring the Motor

These two identical connectors are used to operate the X and Y motors, respectively. They are wired as follows for the BiStepA06 series of controllers (pins counting from top to bottom). The drawings list the BiStepA05 and SimStep units; they are valid for the BiStepA06 as well.



Pin	Name	Description
1	GND	Ground
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	Ground

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

The actual wiring to a stepper motor depends on the motor type. For unipolar, each winding is connected as one side to GND, the other to one of the pins WA-1 to WB-2. For bipolar, the windings match the labels – that is to say, pins 2-3 are for winding B, and 4-5 are for winding A. In reality, the unipolars will also match the labels, but it may be more difficult to identify the windings.

For a unipolar motor, the common line may be identified as distinct from the direct winding lines by its having ½ of the maximum resistance seen between pairs. Note that for a 6 wire 4 phase unipolar motor, there will be two "common" wires; on the you will connect one to pin 1, and the other to pin 6.

For a bipolar motor, if a wiring diagram is not available, an ohm-meter can be used to determine the windings. The low-resistance pairs are the opposite ends of matching windings; high-resistance pairs are different windings.

Once winding lines have been determined, identifying a running sequence can always be done (use clip leads, and turn off power between tests!) via testing the lines using following sequence, connecting to the Y motor:

For wires A, B, C, and D, try

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, ground only J1c-1 (slew Y minus) for a second or so, then ground only J1c-2 (slew Y plus) for a second or so. Only when the correct sequence has been found will the motor actually spin in one direction, then the reverse, for the full second.

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 is the easiest method), 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, we get

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.

Single motor, double current mode of operation

*Starting with GenStepper firmware version 1.59 and above, our products can now be configured to operate a **single** motor at **twice** the rated capacity of the board.*

All of the drivers on our boards can be operated in parallel, in order to double the current available. However, due to timing issues, our updates to each motor are usually offset by 8 microseconds; therefore, in prior firmware releases, you could not simply connect a single motor to the X and Y connectors in parallel, and expect correct operation of the system -- even if you always ran both motors identically. Actually, you would be shorting power to ground!

Starting with GenStepper version 1.59, you can configure the board to send the same signal to the Y motor as is sent to the X motor (with the internal Y operations ignored). If you then wire your motor to BOTH the X and Y connectors (in exact parallel, so that (for example) WA1 from both the X and Y connectors is connected to your Winding A, pin 1 of your motor), then the board can provide double its normal per-winding capacity.

You configure the board to operate this way by GROUNDING both the LY- and LY+ signals. This tells the board that there is no separate Y motor, so it directs the output system to duplicate all X motor signals on the Y connector. You then wire your motor to BOTH the X and Y connectors (as described above); double the current will be available. Please note that if you do not correctly do the above wiring, then you will not get the benefit of the double power mode, and the board is quite likely to fail.

Note that it is also critical that you use just one supply to run both the Vx and Vy motor power when you are using the **double power** mode of operation. Otherwise, you will get problems with "balancing" the exact voltages on the power supplies, since you will be operating them in direct parallel.

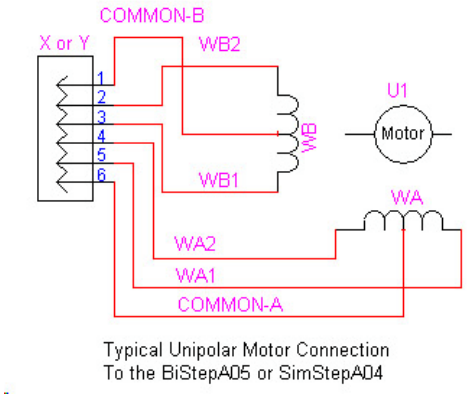
Motor Wiring Examples

The systems have been tested with an interesting mix of stepper motors, both unipolar and bipolar. All were purchased from Jameco (www.jameco.com). The following sections summarize some of the motors tested.

The wiring diagrams shown are labeled for the BiStepA05 and SimStepA04. The BiStepA06 is identical.

Unipolar Motors

This section shows some unipolar motors which were used. Most will work on any of the boards currently available from our company. In each case, the wiring is:



Typical Unipolar Motor Connection
To the BiStepA05 or SimStepA04

Jameco 105873 12 Volt, 0.150 Amp/winding, 3.6 deg/step

This Howard Industries stepping motor has a manufacturing part number of 1-19-4202. It is wired as:

Color	BiStepA06
Black	1
Brown	2
Red	3
Green	4
White	5
<no connection>	6

Jameco 151861 5 Volt, 0.55 Amp/winding, 7.5 deg/step

This Airpax motor has a manufacturing part number of C42M048A04. As with the other Airpax motor, it does not microstep at all. Mode "3o" can smooth its actions, but it does not "stop" at any other points than ½ step locations. It is wired as:

Color	BiStepA06
Green	1
Black	2
Brown	3
Yellow	4
Orange	5
Red	6

When using a 5 volt motor (such as this), you may use a single, 7.5 volt power supply (this may slightly over-voltage the motor), or you may use a split supply. In this case, use a 7.5-12 volt supply for the power to the digital electronics (pins 1 and 4 on the power connector), and a 6 to 7 volt power supply for the motor (pins 2 and 3 on the power connector). The TI driver chips being used drop 1.1 to 2 volts (depending on the chip and the temperature); accordingly, cooling the board becomes quite important, in order to have stable drive voltages for the motor.

Jameco 155432 12 Volt, 0.4 Amp/winding, 2000 g-cm, 1.8 deg/step

This motor provides for 2000 g-cm of holding torque, and has a manufacturing number of GBM 42BYG228. Its wiring order is:

Color	BiStepA06
White	1
Brown	2
Yellow	3
Red	4
Blue	5
Black	6

Jameco 162026 12 Volt, 0.6 Amp/winding, 6000 g-cm, 1.8 deg/step

This motor provides for 6000 g-cm(!) of holding torque, and has a manufacturing number of GBM 57BYGO84. Its wiring order is:

Color	BiStepA06
Black	1
Orange	2
Green	3
Yellow	4
Blue	5
White	6

Jameco 169201 24 Volt, 0.160 Amp/winding, 1.8 deg/step

This excellent motor has a manufacturing part number of STP-57D317. It uses 6 wires, with the wiring being:

Color	BiStepA06
Black (Common lead for PEACH and VIOLET)	1
Peach	2
Violet	3
Yellow	4
Red	5
White (Common lead for Yellow and White)	6

Jameco 173180 12 Volt, 0.060 Amp/winding, 0.09 deg/step geared

This tiny motor has a manufacturing part number of 30BYJ02AH, BF33. Thanks to its gearing, it claims to have both a holding and detent torque of 400 g-cm! It uses 5 wires, already in a connector which directly works with our product. However, two of the wires must be switched (i.e., the order of the wires is incorrect for our use): the pink and yellow wires need to be reversed in the connector. The correct order therefore becomes:

Color	BiStepA06
Red	1
Orange	2
Pink	3
Yellow	4
Blue	5
<no connection>	6

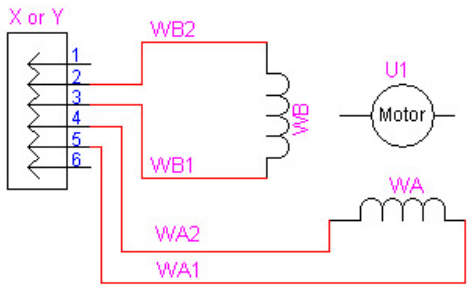
Jameco 174553 12 Volt, 0.6 Amp/winding, 7.5 deg/step

This motor has a manufacturing part number of NMB PM55L-048-NBC7. Its wiring is:

Color	BiStepA06
Black (common for Brown and Red)	1
Brown	2
Red	3
Green	4
Yellow	5
Orange (common for Yellow and Green)	6

Bipolar Motors

This section shows some bipolar motors which were used. They only work on the BiStep products. In each case, the wiring is:



Typical Bipolar Motor Connection
To the BiStepA05 Board

Jameco 117954 5 Volt, 0.8 Amp, 7.5 deg/step

This unit is an Airpax LB82773-M1 2 phase bipolar stepping motor. This motor does NOT microstep at all. It may only be used in full and half step modes (i.e. use the configuration commands "0o", "1o" and "2o")! Mode "o3" may smooth its steps slightly, but it will not really stop at any other than ½ step locations.

When using a 5 volt motor (such as this), you may use a single, 7.5 volt power supply (this may slightly over-voltage the motor), or you may use a split supply. In this case, use a 7.5-12 volt supply for the power to the digital electronics (pins 1 and 4 on the power connector), and a 6 to 7 volt power supply for the motor (pins 2 and 3 on the power connector). The TI driver chips being used drop 1.1 to 2 volts (depending on the chip and the temperature); accordingly, cooling the board becomes quite important, in order to have stable drive voltages for the motor.

The wiring of this unit is therefore:

Color	BiStepA06
<no connection>	1
Yellow	2
Black	3
Red	4
Gray	5
<no connection>	6

Jameco 155459 12 Volt, 0.4 Amp, 2100 g-cm, 1.8 deg/step

This unit is a GBM 42BYG023 stepping motor, which provides for 2100 g-cm of holding torque. It may be wired as:

Color	BiStepA06
<no connection>	1
Brown	2
Orange	3
Yellow	4
Red	5
<no connection>	6

Jameco 163395 8.4 Volt, 0.28 Amp, 0.9 deg/step

This is a Scotts Valley 5017-935 stepper motor. It may be wired as:

Color	BiStepA06
<no connection>	1
Yellow	2
White	3
Blue	4
Red	5
<no connection>	6

Jameco 168831 12 Volt, 1.25 Amp

This motor is a Superior Electric “SLO-SYN” stepping motor, model number SM-200-0050-HL. We ordered it since it stated “1 amp”; however, it turns out to be a 1.25 amp product, and therefore will cause the BiStepA06 to overheat (and probably fail) after just a short period of use, if the BiStepA06 is configured for the default operation of running two motors at a time. We tested it with the wiring of:

Color	BiStepA06
<no connection>	1
White/Brown	2
Brown	3
White/Yellow	4
Yellow	5
<no connection>	6

In order to operate this motor with any of our BiStep units which do not directly handle its current level, you must configure the BiStep to operate in "Single Motor Double Current" Mode. This feature is only available with GenStepper firmware versions 1.59 and later. To do this, you ground the "LY-" and "LY+" input signals (A0 and A1), and you connect the X and Y connectors in parallel to the motor. For example, the "WA1" connection from the Y connector and the "WA1" from the X connector must both be connected to the yellow wire of the motor.

If you fail to wire the unit correctly, you will be shorting power to ground, and are likely to burn up the board! ***This is not a warranted failure!***

Kit Assembly Instructions

The BiStepA06 boards requires average skills to assemble; you must have experience soldering components to printed circuit boards, and you should have an ESD compliant, temperature regulated soldering iron. This manual assumes that you already know how to identify circuit components (resistors, capacitors, and the like) and that you already know how to solder. If you don't know how, we suggest that you purchase one of the pre-assembled versions of the boards, or that you locate someone who can count board-building as one of their skills. Please refer to the **Board Connections** page to view the various board layouts available.

Before you start assembly

- First, make certain that all of the components have been delivered. If there are any missing components, please contact us for a replacement. The following table summarizes all of the part combinations:

Part	BiStepA06
PCB (circuit board)	1
1 K Resistor (Brown, Black, Red)	3
33 K Resistor (Orange, Orange, Orange)	1
0.1 uf cap, 0.1" lead spacing	5
0.1 uf cap, 0.2" lead spacing	1
10 uF electrolytic cap	2
100 uF electrolytic cap	1
LED	1
LM7805T	1
MAX232 IC	1
Ubicom SX28 IC	1
50 MHz ceramic resonator	1
SN754410 IC	2
2 pin header	2
4 pin header	1
3 pin SIP socket	1
28 pin socket	1
16 pin socket	1
DB9F socket	1
4 pin terminal block	1
5 pin terminal block	2
6 pin terminal block	3
Shorting jumper	2

- Note that the integrated circuits are all shipped mounted on an anti-static foam carrier. Please do NOT remove them from that carrier until they are ready to be installed on the board; they are quite sensitive to static discharge, and the parts can be destroyed simply by touching them after walking across a room. The SX28 microprocessor and MAX232 serial controller are not to be installed until the circuit board is completely assembled, to avoid possible destruction of the devices.
- When assembling the board, always take care that you have properly "discharged" any static buildup which you have accumulated before you handle the parts.
- Observe that almost all of the component locations are clearly marked on the printed circuit board (PCB) (as shown in the **Board Connections** section). The only unlabeled parts are the 5 identical 0.1-1.0 uF capacitors, which are positioned around the MAX232 serial driver. They are inserted into the 5 0.1" spaced rectangles which encircle that driver.
- The separate 0.1 uF capacitor with 0.2" lead spacing is inserted near the top of the SX-28 component, into the spot labeled "47p" (we changed the value of the capacitor).
- All of these directions are from the point of view of the component side of the board being face up, with the motor connectors being on the right-hand side (this makes most of the pin-level text be correctly oriented). In this orientation, all of the Pin 1 pins of the vertically mounted IC's (the SX-28 and the two SN754410 units) are located at the top left corner of their respective board locations. The 16 pin MAX232 serial chip is oriented with pin 1 at the lower left corner.
- For all components, the "square" solder pad is considered to be pin 1 for that component or connector.
- For the three 10-100 uF electrolytic capacitors, pin 1 is the "+" side, which is also the side with the longer lead.
- For the LED, pin 1 is the "flat" side (which may be hard to determine), which is also the side with the shorter lead (if there is one). The newer "3/4 size" LED's have equal size leads; on these, the "green" side of the LED is pin 1 (and hence goes to the flat side).
- For the 7805, the metal part of the casing faces the bottom (outside) of the board, as shown by the photo.

Assemble the board

In most small circuit board assembly work, you usually insert the components which have the lowest profile first. By following this rule, you may insert the parts, and then place the board on its back to solder the components, and not have the components fall out. Therefore, you should probably install in the following order:

1. Install the resistors.
2. Install LED diode: See the notes on the prior page for how to tell the polarity.
3. Install the 3 pin SIP socket and the other sockets (28 pin for the SX28, 16 pin for the MAX232. Make very certain that all of the pins make it through the board correctly.
4. Install the 5 identical capacitors around the MAX232, the LED, and the remaining 0.1 uF capacitor (near the top left of the SX28, labeled "47p"). Make certain that the flat part of the LED aligns with the silk screen flat portion.
5. Install the 4 pin SIP header near the top left side of the board, and the two 2-pin SIP headers (SS and JS) near the terminal block connectors on both sides of the boards.
6. Install the electrolytic capacitors (get the polarity right!), and the DB9F connector (make sure no pins are bent!).
7. Install the 6 terminal blocks at their various locations, with the wire-connection sides facing out.
8. Install the 7805
9. Insert both shorting plugs on the 2-pin headers (thus shorting SS and JS)

At this point, you should have a circuit board which can be checked for correct logic circuit power connections. We suggest that you:

10. attach your 7.5-15 volt power supply to the "+Vc" and its associated "GND" connection on the PWR connector (the same way you intend to once the board is complete),
11. apply power. The LED should illuminate. If it does not, **TURN OFF POWER IMMEDIATELY**, and trace your work. Either you do not have adequate power, you applied reverse voltage to the board (i.e., you reversed power and ground, or have the wrong polarity of power supply), you installed the 7805 backwards, you have a bad component (LED or 7805), or you have a short or open circuit.
12. Turn off power, and disconnect the unit from the supply.

Assuming that the above test passed, then you can install the IC components.

13. Carefully install and solder the two SN754410 IC's (remember to take anti-static precautions!). Pin one is at the upper left corner.
14. Install the SX-28 and MAX232 into their respective sockets, avoiding static problems and bent pins. Remember that pin one is at the upper left corner for the SX-28, and the lower-left corner for the MAX232. Note that the SX-28 may have a label which obscures the identity of pin 1. If this is the case, pin 1 will be the lower-left pin, when the label is viewed right side up.
15. Install the 50 Mhz resonator into the 3 pin SIP socket.

You should (hopefully) now have an operational board. Again, apply power, and observe whether the LED illuminates, and the motor blows air down into the heat sink assembly. If either action does not occur, turn the system off, and trace the problem.

If you have an available serial cable, connect a serial port from your computer to the serial connector on the board, and run a standard terminal emulator (such as HyperTerminal) on your computer. Set the communications to 9600 baud, no parity, and 1 stop bit. Just turning on power to the board should be enough for it to send a "sign on" message to the computer (this is the copyright message from the sx-28 chip).

If you have one of the stepper motors described in the **Motor Wiring Examples** section, you should be able to see it run by wiring it as shown, connecting it to the Y motor connector, and then grounding the Y- pin of J1c. You may need to remove the SS (Single Supply) shorting jumper, and provide split power, if your motor exceeds the 7.5-15 voltage range required by the Vc power input.