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 BS1010, BS1010W and SS1010 Stepper Motor Controllers

By
Peter Norberg Consulting, Inc.
Matches GenStepper Firmware Revision 5.51

Table Of Contents

Disclaimer and Revision History

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

This manual describes artworks BS1010, BS1010W and SS1010. The firmware release described is GenStepper version 5.51. 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
5.28	August 14, 2011	Upgraded to new assembler and linker; improves detection of illegal interrupts. Enabled automatic firmware support for use of external precision oscillator as clock source.
5.30	February 19, 2012	Enhanced serial buffering, to allow for fully buffered I/O (see the 'V' command)
5.40	November 20, 2012	Extensively modified 'step-and-seek' mode to improve its behaviors when incoming pulse rates are less than the target run rate
5.43	April 23, 2013	Correct error in the "regoto" functionality. If you issue a 'z' command on a targeted goto ('G') command, and then re-issue a 'G' command to the same location, the second 'G' command could be ignored.
5.46	August 21, 2013	Added 't' configuration option to control whether the first step on a new motion request occurs 'instantly' or after a delay of one step cycle. By default, it operates in the delay mode, for compatibility with existing applications. **This update invalidates any saved settings.**
5.47	August 23, 2013	Added capability of user specifying sense level (high or low) for the NXT-based instant-stop operation (controlled by the 't' command).
	September 5, 2013	Corrected a/d input command documentation
5.48	October 14, 2013	Added optional separate ramp rate for use during a limit event (extension to 'P' command)
5.51	June 8, 2014	General bug fixes, correct 'L' documentation to include 'motor is blocked from motion' bit

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 BS1010 and SS1010 series of motor controllers have components that can get hot enough to burn skin if touched, depending on the voltages and currents used in a given application. Care must always be taken when handling the product to avoid touching these components:

- The 2940 5 volt regulator (located directly beside the USB serial connector, next to a tall electrolytic capacitor)
- On the BS1010 series, the two SN754410 power drivers (both located near the center of the board)
- On the BS1010 series, the PCB board under the SN754410 power drivers and under the 2940 regulator

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 only protected against static electricity when it is correctly connected to a grounded power supply, with the GND power input truly connected to earth ground. 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.

The board must be adequately cooled based on your intended use -- please see the separate section describing board mounting and cooling (Board mounting and cooling considerations).

LIFE SUPPORT POLICY

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

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

Introduction and Product Summary

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

The **BS1010** and **SS1010** microstepping motor controller from Peter Norberg Consulting, Inc., have the following general performance specifications:

	BS1010	**BS1010W**	**SS1010**
Unipolar Motor	Yes	Yes	Yes
Bipolar Motor	Yes	Yes	No
Maximum Motor supply voltage (Vx and Vy)	34V	15V	26V
Maximum Logic supply voltage (Vc)	15V	15V	15V
Quiescent current (all windings off)	150 mA	150mA	150mA
Maximum winding current (per motor winding, requires external fan to operate)	1.0A	1.0A	1.0A
Board size, BS1010	2.25" x 3.0"	2.4" x 2.5"	1.75" x 3.0"
Dual power supply capable	Yes	No	Yes

Each board can be controlled simultaneously via its TTL input lines and its USB serial interface (the BS1010W does not have USB, however). 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 57,600 microsteps per second), slope rates (1 to 57,600 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 57,600 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 allow for user upgrades of the firmware via the USB interface, by running an update application which is available from the Stepperboard.com web site.

Notes on the BS1010 and SS1010 USB Power and Ground Isolation

The BS1010 and SS1010 artworks optionally provide for some ESD protection to most of the TTL input signals (except for the RST 'reset' input line), as well as for either optional isolation of the USB signals from the rest of the board (to avoid potential ground loops), or for optional use of the USB cable as a power source for the logic portion of the board (NOT to run motors).

This ESD protection is not available on the current BS1010W artwork.

BS1010 artwork

SS1010 artwork

The ESD protection consists of special protection diodes connected between most of the TTL input signals and the board ground. This means that the protection only works if the board ground (the GND signals on the PWR connector) is connected to a power supply that can absorb the ESD event. If the board is not connected to a power supply, or if the supply does not redirect the GND to real earth ground, then the board ESD protection may not be adequate.

The signals that are ESD protected are:

LIM: LY-, LY+, LX-, LX+

SLEW: Y-, Y+, X-, X+

IO: NXT, RDY, SI, SO, AN2, SI2, SO2, IO2

USB: All signals

Optionally, the USB signals can be digitally isolated from the rest of the board (order option). This means that there is no longer a potential for a ground loop to occur between the power supplies that you use to power the motors and the computer – they are fully isolated from each other.

Alternatively, you may request that the logic portion of the board be capable of being powered off of the USB cable. In this case, the USB cannot be digitally isolated; since power and ground for the board comes from the USB system, isolation is not meaningful.

Short Feature Summary

- One or two stepper motors may be independently controlled at one time.
- Each motor may be either Unipolar or Bipolar (Unpolar only for the SS1010).
- Each motor may draw up to 1.0 amps/winding (0.5 to 1 amp, depending on order options, for the SS1010). Note that an external cooling fan ***must be used*** when your motor draw exceeds 0.6 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 BS1010 1 amp product can operate a single 2 amp motor, when this feature is enabled (assuming that the board is adequately cooled). On the SS1010 series, the 0.5 amp version fully supports double power mode; however, the 1 amp version only supports this mode to the extent that it allows for simultaneous identical motion between motors.
- Limit switches may optionally be used to automatically request motion stop of either motor in either direction.
- Rates of 1 to 57,600 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 > 1,600,000,000 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 normally 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 USB serial connection.
- An additional mode is available which allows an external computer to directly generate step sequences on the motor control lines. Up to 57,600 steps per second may be requested.
- Runs off of a single user-provided 6.5 to 15 volt DC power supply, or two supplies (6.5-15V for the logic circuits and 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.
- Direct support on board for 'daisy chaining' boards using a subset of the SerRoute system
- 3 of the pins may be individually reprogrammed as 12 bit A/D inputs, TTL inputs, or as TTL outputs. Additionally, you may reprogram one or two of those pins to be used as voltage-based rate controls for the motors (such as using a potentiometer to set the rate)

- All of the slew inputs may be reprogrammed for use as generic TTL inputs
- On the BS1010, the AN2, SI2, SO2 and IO2 pins may be fully reprogrammed for use as generic digital I/O pins. The BS1010W does not support these extra pins.
- Optionally, a precision oscillator may be used to allow for exact rates

Default Firmware Configuration: Power-On (and reset) Defaults

All of the default settings on the board (such as microstep size, ramp rates and so on) may be reprogrammed by you to be remembered after a power cycle event (see the 'e' command for EEProm operations). However, all of those features have a 'factory default' value which is used whenever an EEProm reset is done or when a 'reboot using factory default' command is given ("-1!").

- **4!** – Set the microstep size to 1/16 (4/64ths) of a step per logical step.
- **3072A** – Set the Automatic Full Step rate to be >=3072 microsteps/second
- **B** – Select both motors for the following actions
- **9600b** – Set the communications baud rate to 9600 baud
- **0E** – Operate in normal (non-step-and-direction) mode
- **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** – (three-oh) 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 motor to 800 microsteps/second
- **'r' settings** – Set the POT rate scales such that 0 volts maps into a rate of 1, and 5 volts maps into a rate of 57,600 for both motors
- **0T** – Enable all limit switch detection
- **4065t** – Set all I/O ports to our defined default settings
- **1V** – Set <CR><LF> sent at start of new command, no transmission delay time, normal 29.5 MHz system processor speed.
- **0W** – Full power to motor windings

Cooling Requirements

If you are operating motors that require more than 600 mA of current per winding on the BS1010 (or 200 mA per winding on the SS1010), or if your motor voltage exceeds 15 volts then you must provide for fan-based cooling of the board. We suggest at least 8-10 CFM, directed either across the top of the board, or downward towards the board (so that both the 2940 and the driver chips are in the direct path of the airflow). You may need to provide fan cooling even if the current is less than 900 mA, depending on the ventilation you provide in your enclosure.

If you intend to leave the power windings on (via the variations of the 'W' command), or if you are not providing for any air flow in your enclosure (i.e., you are using a sealed box), then you are very likely to have to cool the board, even if the current is under 600 mA (or 200 mA on the SS1010)/winding.

USB Driver Installation Under Windows for the BS1010 and SS1010

The BS1010 and SS1010 boards (not the BS1010W) use a USB driver chip for communications with your hosting computer. FTDI (http://www.ftdichip.com) provides drivers for operation under Windows[tm], Linux, and Mac/OS.

For use under Windows XP and Vista, our installation disk includes modified copies of their Windows[tm] drivers; our boards normally are shipping using a unique ID code which prevents them from being recognized by the default FTDI drivers.

All Linux and Mac/OS customers must download their drivers directly from the http://www.ftdichip.com site, as we have no support capability for those platforms. They will also have to special-order the boards from us such that we configure them to respond to the standard FTDI drivers. Look for the drivers and documentation that relate to their FT232RL device.

A short summary of the installation of the drivers under Windows XP or Vista follows. For installation under Linux or on the Mac, please refer to the FTDI documentation available from their web site.

Base Driver Installation Under Windows XP, Vista and Windows 7

With the new drivers (as of December 5, 2009), the installation of the USB drivers has been heavily streamlined and simplified. Normally, the installation is done as one of the intermediate steps of installing the documentation package (from the SetUpStepperBoard.exe installation application provided on our CD or as a download from our web site). If you elect to not do the installation at that time, the actual current version of the installer is installed in your programs menu, under:

Start->Programs->StepperBoard->USB Support Tools->Install FTDIStepperBoard USB Drivers

This tool requires that you are logged on as an administrator: it will not correctly install the drivers if you are a 'standard user'. The process normally is:

1. If you have not already done so, log on as an administrator to your computer. If you are not logged on as an administrator, then the following procedure will be blocked as soon as you attempt to install the drivers!
2. Make sure that none of our boards are plugged in.
3. Run the installer to completion, and exit any remaining "completion" panels.
4. Insert the small square end of the A-B USB cable into the stepper motor control board. Insert the large flat end into a free USB port into your computer. **Please make certain that the cable fully "snaps" into our connector – this can take a noticeable amount of force**. If it is not properly "seated", then Windows will not correctly recognize the board.
5. Once the installation process completes, the code will automatically add a new "COM" serial port which is "attached" to the board when it is plugged into the same USB port on your computer. *The system will automatically add a new COM port each time you attach a new board to any other USB port on your computer or hub. It may also create a new COM port if you receive a repaired board back from us (if we have had to replace the USB driver chip).*

Initial testing of the board after driver installation – TestSerialPorts

The easiest way to test the board (and to identify which COM port is being used for board communications) is to run our "TestSerialPorts" application (found under 'StepperBoard' on your 'Start' menu). This application will scan all of the potential COM ports on your system (from COM1 through COM255), and will identify every port that has a connected StepperBoard product powered and attached.

The test assumes that the board is correctly powered in order to 'talk' to the com port: in the case of the AR-BS1010 board, you either have to correctly power the board with your own power supply (as described under power connectors), or you have to configure the power jumper on the board to the 'USB' power position in order to allow the logic portion of the board to be powered off of the USB cable (this is an optional feature and may not be present on your particular board).

When TestSerialPorts starts, simply press the "Scan Serial Ports" button (you may safely ignore the other buttons). The application will then perform its scan, and will identify every COM port on your system. It will also attempt to identify the baud rate for each connected board, assuming that the board is set to a baud rate that it recognizes.

If TestSerialPorts does not locate your board, please contact us for additional tests to perform. Remember that the board must be connected to your computer and powered on, and the FTDI USB drivers must be correctly installed for TestSerialPorts to be able to locate the board.

Please note that the TestSerialPorts application will locate our board even if you have not adjusted the default USB COM port properties, as described in the next section.

Adjusting Default COM port properties for best operation

Once the system has created the COM port for the board, you may need to change the system defaults to match the requirements of our motor controllers. If you installed from the 'FtdiStepperBoard' subdirectory, then these changes will normally have been done for you. Otherwise, you will need to perform the following procedure:

1. Under Windows 2000 or XP, go to your "system properties" page. Do this by
    a. Right-click on your "System" icon
    b. Select "Properties"
    c. Select "Hardware devices" (it might just be called "Hardware")
    d. Select "Device Manager"
2. Under Windows Vista and Windows 7, log in as an administrator, and then get to your "device manager" page by:
    a. Go to your 'Start' menu, and click on the 'Computer' button
    b. On the ribbon that appears at the top of the resulting window, click on "System Properties"
    c. On the task pane on the left of the new window, click on "Device Manager"
    d. The system may ask for your permission to continue. Press the "Continue" button.
3. Look under "Ports (COM and LPT)", and select the COM port that you just added (it will normally be the highest-numbered port on the system, such as "COM6"), and edit its properties. Note that the 'TestSerialPorts' application (described in the prior section) will have identified this COM port for you as part of its report.
4. Reset the default communication rate to:
    a. 9600 Baud,
    b. No Parity,
    c. 1 Stop Bit,
    d. 8 Data Bits,
    e. No Handshake
5. Select the "Advanced Properties" page, and set the:
    a. Read and Write buffer sizes to 64 (from their default of 4096).
    b. Latency Timer to 1 millisecond
    c. Minimum Read Timeout to 0
    d. Minimum Write Timeout to 0
    e. Serial Enumerator to checked
    f. Serial Printer to unchecked
    g. Cancel If Power Off to unchecked
    h. Event On Surprise Removal to unchecked
    i. Set RTS On Close to unchecked

(that is to say, only the Serial Enumerator is checked in the set of check boxes on the display)

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 USB 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

All TTL input signals are treated as TTL inputs using Schmitt-triggering. This means that a logic "0" is generated at any time that the input voltage drops to <= 1 volt, and a logic "1" is generated when the input voltage is above 4 volts.

Note also that the LIM and SLEW TTL inputs are tied to +5 via a 1K resistor. 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. Unless you special-order the board from us without the pullup installed, you may rely on this resistance value as being valid for a current-based driver.

For the extended TTL signals of RDY, NXT, AN2, SIO2, SO2 and IO2, there are no pull-ups or similar signal conditioners mounted on the board. All of these signals can be programmed as being either inputs or outputs, and adding board-based pull-ups would cause unnecessary heating of the microprocessor. This means that for any of those signals which are used as inputs, you must either provide a true voltage source, or you must actively tie the signal to our +5V using an appropriate resistor (1 to 10 K ohm).

Input Limit Sensors, lines LY- to LX+

Lines LY- through LX+ are normally 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, the firmware also supports 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

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 1K resistor.

The stop requested by a limit switch normally is "soft"; that is to say, the motor will start ramping down to a stop once the limit is reached – it will **not** stop instantly at the limit point (unless a special firmware option is ordered). You may use the 't' command to reconfigure this behavior to be an instant stop, if that is needed.

Under the 'soft' stop, if a very slow ramp rate is selected (such as changing the speed at only 1 microstep per second per second), it can take a very large number of steps to stop in extreme circumstances. It is quite important to know the distance (in microsteps) between limit switch actuation and the hard mechanical limit of each motorized axis, and to select the rate of stepping ("R"), rate of changing rates (the slope, "P"), and the stop rate ("K") appropriately.

As the most extreme example possible:

- if for some insane reason the motor is currently running at its maximum rate of 57,600 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 57,600 seconds (about 16 hours -- groan!), with a distance of ½ v^2, or ½ (57,600)^2, or 1,658,880,000 microsteps.
- Note that this same amount of time would have been needed to get up to the 57,600 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!***

If you configure the system for 'instant stop' using the 't' command, ***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 slew inputs are designed to operate via a microswitch closure to ground. NXT does not have a pullup resistor, so it requires a true voltage source if it is used in its optional 'Next Rate' configuration.

The connections are:

Signal	Action Requested
Y-	-Y
Y+	+Y
X-	-X
X+	+X
NXT	Rate Control (configurable)
RDY	Motors Ready, or rate control (configurable)

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 'NXT' input can be configured by you to accept a TTL input signal to request a new rate (using the 't' command). 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

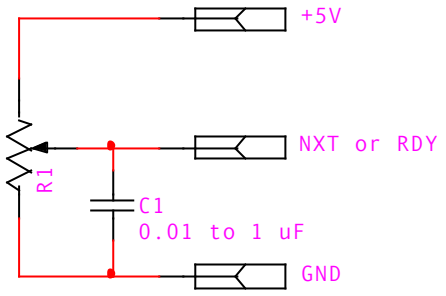
Be forewarned that there is no way for the software to tell that a motor cannot operate at a given rate. On power-on, the default microstep is 1/16th of a full step; therefore, the default rates range from 1 to 500 full steps/second. Changing the microstep size does change the above real "full step"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

The RDY output signal is normally configured by you 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.

As a set of extended operations, both NXT and RDY can be configured by you to accept a 0-5 volt analog input signal, which can be used by the firmware to select the rate for one or both motors. See the 't' command for more details.

Potentiometer Rate Control

Through use of your own 1K potentiometer (or pair of potentiometers), you may optionally control the rates of the motors by setting a pot. For both the NXT and RDY inputs (if selected by you through use of the 't' command), the schematic appears as follows (note: R1 is a 1K potentiometer):



Variable Resistor Circuit for use with GenStepper firmware on the BS1010 board

All of the board connections (+5V, NXT, RDY and GND) are available on the IO connector on the BS1010. Capacitor C1 is optional; it may be needed if you get too much variance in the rates generated by the circuit (due to noise on the analog input data).

Once you have wired the above circuit to the board, you may use the 't' command to tell the board that the NXT and/or POT inputs are to be treated as analog sources of rate for the X and/or Y motors (the firmware supports complete flexibility here: either input may control either or both motors). If you do not use the 'r' command to set the rate scales, then the system is configured such that 0 volts maps into 1 microstep/second, and 5 volts (A/D value 4095) maps into the maximum supported board rate of 57,600 microsteps/second. You may use the 'r' command to independently set the scalings of the analog inputs for each motor, with the added flexibility of allowing the values to be inverted (for example: you may map 5 volts into a rate of 1, while mapping 0 volts into a rate of 20,000, if this works better in your application).

Please be forewarned: there is only marginal protection in the system for a broken wire on the analog input. The code attempts to see if the input is flying around by performing a simple least-squares calculation: however, it still can misinterpret a disconnected wire as being valid, thus giving you random rates.

Analog Input Signals

Three of the input signals (NXT, RDY and optional AN2) may be configured through use of the 't' command to act as analog inputs. When they are configured this way, they accept 0 to 5 volt positive voltages, which map into A/D readings of 0 to 4095 (respectively).

The input system requires less than 2.5K of input impedance: this means that your signal sources need to be somewhat 'hard' (higher current), otherwise you may get more noise on the input than you would prefer.

You may also find that you get better results by placing a small capacitor (0.01 to 1 uF) between the selected input (such as NXT) and GND, as a simple filter for the input. If you do this, you will get better results having the capacitor being close to the actual input connector than if the capacitor is at your voltage source, since it will help to reduce induced noise picked up by your wires running from your signal source to the board.

You use the 'a' command to read the signals, as is described here.

Serial Operation

The USB serial control of the system allows for full access to all internal features of the system. It operates at almost any legal baud rate from 2400 to 115,200, 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. You may use the 'b' command to select the baud rate: it will accept almost any rate, although only a few will be 'perfect'. The suggested rates to use would be 2400, 4800, 9600, 19200, 38400, 57600 or 115000.

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 upon completion of receipt of the stop bit of the received character. This means that the command normally starts being processed within 16 microseconds of 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 sending processor is ready for it! In microprocessor, non-buffering designs (such as with the Parallax, Inc.[tm] Basic Stamp [tm] series of boards), this can be a significant issue. The firmware handles this via a configurable option in the 'V' command. If enabled, the code will pause for about 1 millisecond after receipt of a new command character; for the Basic Stamp[tm] this is quite sufficient for it to switch from send mode to receive mode.

Routing Serial to 'Child' Boards

The BS1010 and SS1010 series of boards support optional "Serial Routing" of data to and from other boards via the "SI2" and "SO2" TTL-Serial connectors. This permits "daisy-chaining" of the boards, so that more than 2 motors may be easily operated off of one serial port. Additionally, GenStepper version 5.0 and later supports "binary" serial transmissions – that is to say, it allows communication with a "child" board which includes non-visible characters, thus permitting communication with other products not supplied by Peter Norberg Consulting, Inc. This routing behavior will not work on the BS1010W, since the signals are not available.

Connecting your 'Child Board'

The SI2 and SO2 signals on the BS1010 and SS1010 are normally used as the TTL-Serial connections to the child board. These are NOT RS232-level signals; instead, they are standard TTL signals, 0 to 5 volt in value. The signals are:

- SI2: TTL-Serial Input received from child board
- SO2: TTL-Serial Output sent to child board

You also need to make certain that both boards have the same ground potential: they both need to have their logic supplies run off of the same power supply. Ground should be done via a "star" ground, which means that all grounds are tied to a common point (NOT "daisy-chained"!).

Routing the Serial Data

At power on, the board is configured to directly interpret and execute all incoming serial data, and to leave the SI2/SO2 serial echo lines idle. This mode means that the board is under control of the serial data stream from the host. There are two methods of exiting this mode, and switching to serial-routing operation (wherein data is routed from the serial port to and from the SI2/SO2 pins as needed).

One method, the "Binary Route" technique, allows single-character entry into route mode, and is the preferred method when controlling only one child board or when controlling a child board which is not provided by Peter Norberg Consulting, Inc. (and therefore does not match our normal communications protocol).

The other method, the "SerRoute compatible" technique, allows the BS1010 or SS1010 board to operate as both a router and a motor controller within a subset of the rules of the "SerRoute" system (see the separate "SerRoute" manual for complete SerRoute specifications – GenStepper version 5.0 supports just the '{xxx}' style of routing).

'<' – Start Binary Route

The '<' character immediately starts routing all serial data to and from the child board. Except for the route command characters ('<', '>', '{', '}') and the ESC character ('\'), all characters from the serial port are sent unchanged to the SO2 child serial port without any other interpretation by the firmware.

This mode is the preferred method when operating any board as a child that is not a Peter Norberg Consulting, Inc. product. It permits transmission of binary data, and avoids any issues of sending spurious route-related commands (such as the '{' "start route selection" character).

Please note that, when routing in the binary route mode, you must precede certain characters with the escape character '\' in order to prevent the firmware from interpreting those characters as route commands. This means that you should precede any of '<', '>', '{', '}', or '\' with a '\' ("escape it"), so that it will be passed on unchanged to the child board.

'>' – Stop Routing

The '>' character immediately stops any routing which is taking place. This is normally used to exit the routing mode, and return to "talking" to the main board.

Note that if the '>' character is preceded by the '\' escape character, the '>' is just sent on to the child (assuming that routing is enabled), and route mode is not canceled.

'{xxx}' – Select SerRoute compatible Route Mode

This sequence allows routing using a method which is compatible with the SerRoute firmware. The 'xxx' values are used to select which "route" is to be enabled, while the '{' and '}' are used to bracket the route address. Please see the 'SerRoute' manual for a full explanation of the routing rules: from the point of view of BS1010, however, there are only 3 important SerRoute-compatible route modes, as selected by the first character in the 'xxx' value.

x	Description
Empty	If just '{}' is sent, then the current board is selected for all serial. This is exactly equivalent to receipt of the '>' character, above.
'0'-'8'	All serial data is routed to pin SO2, and all serial data seen on pin SI2 is routed back to the host.
'9'	Both this board and the child board receive all following serial data, while the serial data sent back to the host is strictly that from this board. Used to execute commands simultaneously on both boards.

In reality, when interpreting the '{xxx}' style of input, the code operates as follows:

1. When the '{' is seen, the code sets a flag for 'the next character tells us how to route all of the following characters'.
2. When the first 'x' character is seen, if it is not '}' or '0'-'9', the route command is cancelled (illegal sequence), and the 'x' character is interpreted as a new board command. I.e., '{g' is interpreted as 'g' (although technically speaking, this form of command is not proper).
3. If the first 'x' character is '}', then routing is terminated. All following characters are treated as normal data to the board, and no further activity occurs relative to the IO6/IO7 ports (aside from completing any pending serial transmission).
4. If the first 'x' character is '0' to '8', then all future serial data is routed to the child board via the SI2 and SO2 lines. The '{' "route selection" character is passed on to the child, to allow for nested route control.

5. If the first 'x' character is '9', then (as with '0' to '8') all future serial data is replicated to the IO7 output line. However, it is also executed locally, thus placing the board into the mode of 'broadcast'. The '{' "route selection" character is passed on to the child, to allow for nested route control.

For example, if we have 3 BS1010 boards with GenStepper firmware "daisy chained" such that:

- the first board "talks" via its USB port to the host,
- the second board has its SI line connected to the first board's SO2 line, and its SO line to the first board's SI2 line, and
- the third board has its SI line connected to the second board's SO2 line, and its SO line to the second board's SI2 line, giving us:

First Board

→ Second Board

→ Third Board

We would address the first board with:

{}

The second with:

{0}

The third with:

{00}

And all three at once with:

{99}

Please refer to the "SerRoute" manual for many more examples of this technique.

Serial Commands

The serial commands for the system are described in the following sections. The code is case-sensitive, but will treat any lower-case item as its upper-case equivalent if there is no overriding lower case version of the command (i.e., "s" means the same thing as "S", but "t" has a different meaning from "T"). ***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 preceded with a number, to set the value for the command. If no value is given, then the last value seen is used.

All commands are to be treated as being case sensitive! Although some commands may be insensitive to case, it is far safer to treat all commands case sensitive. Most commands are to be upper case – the lower-case versions are extended commands that are not available in earlier GenStepper-based (pre version 5.0) product offerings.

General Commands

- 0-9, +, - – Generate a new VALUE as the parameter for all FOLLOWING commands
- b – Select the communications baud rate
- L – Latch Report: Report current latches, reset latches to 0
- V – Verbose mode command synchronization and processor clock control
- ! – RESET – all values cleared, all motors set to "free", redefine microstep. Duplicates Power-On Conditions!
- ? – Report status

Motor Control Configuration

- A – Select the Auto-Full Power Step Rate
- E – Enable or Disable Remote Direct Pulse Control
- e – Write EEProm data
- H – Operate motors at ½ power
- O – step mOde – How to update the motor windings
- T – limiT switch control
- r – Set rate scaling parameters for X, Y motor rate control
- W – Set windings power levels on/off mode for selected motor

Motor Selection

- B – Select both motors
- X – Select motor X
- Y – Select motor Y

Motor Motion Configuration

- K –Set the "Stop oK" rate
- P – sloPe (number of steps/second that rate may change)
- R – Set run Rate target speed for selected motor(s)

Motor Motion Control

- G – Go to position x on the current motor(s)
- I – Wait for motor 'Idle'
- M – Mark location, or go to marked location
- S – start Slew
- Z – Stop current motor
- = – Define current position for the current motor to be 'x', stop the motor

TTL and Analog I/O

- a – Read A/D channel
- C – Set TTL Lines high that are '1' in 'x'
- D – Set TTL lines to requested values
- t – Configure all I/O port directions and use
- U – Set TTL Lines low that are '1' in 'x'

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. This mode will also disable ½ current mode ("1H") once this rate has been reached.

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. When operating at the default microstep resolution of 1/16th step size, then the 3072 rate maps into 192 full steps/second. When operating at a microstep resolution of 1/64th 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 57600 to disable this feature.

a – Read A/D channel

This command allows you to read one of the three available 12 bit A/D input channels, as requested by the parameter (x). The given input must already be configured as an analog input through use of the 't' command in order for this function to not report a 0 result.

The firmware takes 16 samples on the requested channel, and reports the average of those samples.

```
0a – RDY, when configured as an analog input
1a – NXT, when configured as an analog input
2a – AN2, when configured as an analog input
```

The report consists of data in the following form:

a,#CH,#AD

where:

'a' is the report type; in this case, 'analog data report'

'#CH' is the channel being reported

'#AD' is the actual 12 bit (0 to 4095) reading for the channel

For example,

0a

Could report

a,0,345

which would mean 'the signal on RDY had an A/D value of 345.

A reading of 4095 maps into 5 volts.

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.

b – Select the communications baud rate

The '**b**' command is used to change the communications baud rate from its current value to a requested new value. The factory default setting is 9600 baud, or

9600b

The code takes the rate that you request, and calculates the closest rate to that which it is capable of supporting. It then echoes that rate back, in a response formatted as:

b,#

where '#' is the real baud rate that the code will be able to use. After it completes sending of the final '*' in its response, it will then reset itself to operate at the newly specified rate.

For example, to set the firmware to communicate at 19,200 baud, you would send:

19200b

and you would get the response:

b,19200

*

After you receive the '*', you would then reset your serial system to operate at 19,200 baud.

Suggested values for the baud (which are used exactly) are:

```
2400
4800
9600
19200
38400
57600
115200
```

Many other rates are supported; however, the above list contains the most commonly used values.

Please be forewarned! If you use a non-standard value, or a value above 19,200, then our Stepperboard class libraries (both the Active-X and the Visual Studio .NET systems) may not be able to 'find' the board using an automatic scan. You will need to directly tell the class library what baud rate and what port to use for your application.

If you attempt to set the baud rate to an unsupported value, and ignore the response which contains the real value used, you may not be able to talk to the board. Simply power it off and back on: it will revert to the baud rate that was active the last time that you performed the special '123456789e' save state command, or to the 9600 baud factory default if you have never saved the firmware state. If you have saved a non-standard baud rate using the save-state command and you forget what you used as the baud rate, you will have to perform the factory-reset action, as described elsewhere in this manual, in order to revert the firmware to a known standard state.

C – Set TTL Lines high that are '1' in 'x'

'C' is used to set selected bits high on any of the TTL ports that are enabled as TTL outputs. It is bit encoded as:

Bit	Hex Value	Decimal Value	Description
0	0x0001	1	Y WA1, if Y drivers enabled (see bit 9)
1	0x0002	2	Y WA2, if Y drivers enabled (see bit 9)
2	0x0004	4	Y WB1, if Y drivers enabled (see bit 9)
3	0x0008	8	Y WB2, if Y drivers enabled (see bit 9)
4	0x0010	16	X WA1, if X drivers enabled (see bit 8)
5	0x0020	32	X WA2, if X drivers enabled (see bit 8)
6	0x0040	64	X WB1, if X drivers enabled (see bit 8)
7	0x0080	128	X WB2, if X drivers enabled (see bit 8)
8	0x0100	256	Enable X driver outputs, if enabled for this command via the 'T' command
9	0x0200	512	Enable Y driver outputs, if enabled for this command via the 'T' command
10	0x0400	1024	RDY
11	0x0800	2048	NXT
12	0x1000	4096	AN2
13	0x2000	8192	SO2
14	0x4000	16384	SI2
15	0x8000	32768	IO2

Add the decimal value for each bit that you want to set to '1' (high), and issue the command on the sum. Those bits will get set: the others will be unchanged.

Note that you need to use the 'T' command (T – limit, slew switch control and motor driver enable) in order to enable access to the motor output drivers, and the 't' command in order to enable access to the other TTL I/O lines. Note also that if you are using the motor driver outputs (from the 'X' or 'Y' connectors), you must have the given output enabled in order for it to actually drive the output high or low (see bits 8 and 9).

For example, to set the IO2 line high, you could issue the command:

32768C

(assuming that IO2 was configured as a TTL output).

There are two other commands which may be used to control the output port levels:

- D – Set the TTL lines to the requested values
- U – Set the TTL lines low that are '1' in 'x'
- Note that the current input values for the limit and slew switches can be retrieved with the '5?' (limit switch) and '6?' (slews plus extras) commands, as described in the Other report values section of this manual.

D – Set TTL lines to requested values

'D' is used to set all of the TTL signals are enabled as TTL outputs to your desired values. It is bit encoded as:

Bit	Hex Value	Decimal Value	Description
0	0x0001	1	Y WA1, if Y drivers enabled (see bit 9)
1	0x0002	2	Y WA2, if Y drivers enabled (see bit 9)
2	0x0004	4	Y WB1, if Y drivers enabled (see bit 9)
3	0x0008	8	Y WB2, if Y drivers enabled (see bit 9)
4	0x0010	16	X WA1, if X drivers enabled (see bit 8)
5	0x0020	32	X WA2, if X drivers enabled (see bit 8)
6	0x0040	64	X WB1, if X drivers enabled (see bit 8)
7	0x0080	128	X WB2, if X drivers enabled (see bit 8)
8	0x0100	256	Enable X driver outputs, if enabled for this command via the 'T' command
9	0x0200	512	Enable Y driver outputs, if enabled for this command via the 'T' command
10	0x0400	1024	RDY
11	0x0800	2048	NXT
12	0x1000	4096	AN2
13	0x2000	8192	SO2
14	0x4000	16384	SI2
15	0x8000	32768	IO2

Add the decimal value for each bit that you want to set to '1' (high), and issue the command on the sum. Those bits will get set: the others will be unchanged.

Note that you need to use the 'T' command (T – limit, slew switch control and motor driver enable) in order to enable access to the motor output drivers, and the 't' command in order to enable access to the other TTL I/O lines. Note also that if you are using the motor driver outputs (from the 'X' or 'Y' connectors), you must have the given output enabled in order for it to actually drive the output high or low (see bits 8 and 9).

For example, to set the IO2 and RDY lines high, and everything else low, you could issue the command:

33792D

(assuming that IO2 and RDY are configured as TTL outputs).

There are two other commands which may be used to control the output port levels:

- C – Set the TTL lines high that are '1' in 'x'
- U – Set the TTL lines low that are '1' in 'x'
- Note that the current input values for the limit and slew switches can be retrieved with the '5?' (limit switch) and '6?' (slews plus extras) commands, as described in the Other report values section of this manual.

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.

As of version 5.23 of the firmware, 'E' is ignored if you fail to provide a parameter (i.e., 'E' alone results in the board ignoring the command, while '0E' causes the normal 'E' operation with a parameter of '0'). The options are bit encoded into the value; the low 2 bits of value define the main Pulse Control mode, the next 2 bits are extended feature selection, while the next higher 4 bits control the interpretation of the input signal level.

The bits are encoded as follows:

Bit	Value	Description
0-1	+0 to +2	Pulse mode to use: 0 means disable, 1 means each line is own step in its own direction, 2 means step-and-direction mode
2	+4	TTL control of motor current
3	+8	Enable 'Step-And-Seek' mode (firmware 5.05 or later). Disables bit 2 (TTL control of motor current)
4	+16	Invert Y-
5	+32	Invert Y+
6	+64	Invert X-
7	+128	Invert X+

The defined values for the low 2 bits (0-1) are:

0E – Disable remote pulse control (the power on/reset default). Note that the *entire* value must be 0 for remote pulse control to be disabled – if you define any of bits 2-7 to be non-0, then the code will act as if step mode '1' was selected.

1E – Enable remote pulse control, with each line being its own step/direction

2E – Enable "Step/Direction" mode of direct pulse control: the "-" inputs are treated as direction, the "+" inputs are treated as step requests.

Bit 2 (add 4 to the above command) is used to define whether the limit switch inputs are used to control the current to the motors. If bit 2 is set (+4), then this extended TTL control of the motor current is enabled as described later in this section. If bit 2 is clear (+0), then the limit switches are either ignored or are used as true limits, depending on whether the "hard stop" option is enabled with the 't' command.

Bit 3 (add 8 to the above command) is used to enable 'Step-And-Seek' mode (firmware 5.05 or later). . Since "Step-And-Seek" requires normal use of the LIMIT inputs, setting this bit also disables bit 2 (TTL control of motor current via the LIMIT inputs).

Bits 4-7 are used to control the interpretation of the signal levels. When set to 0 (the default), then the signals are interpreted as described below. When set to 1, then the given signal is inverted (i.e., "0" is mapped into "1", and "1" is mapped into "0"). The net effect of this is to change the edge which triggers the motion from the low-going edge to the high-going edge, and to flip (when in mode 2) the interpretation of the direction of travel.

For example, to operate in the Step/Direction mode of operation, with high-going pulses requesting the steps on both the X and Y motors, you would use a value of 2+32+128 (since the Y+ and X+ input signals are used as the step requests, and need to be inverted so that the high-going edge triggers). Therefore, the command given would be "**162E**".

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.

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, UNLESS THE 'INSTANT STOP' LIMIT MODE IS ENABLED OR THE 'STEP+SLEW' FEATURE IS USED.**

When step-and-direction mode is 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). If "step+slew" mode is enabled, the 'G' and 'S' commands still work; extra step pulses simply amend the motion request.

This mode monitors the TTL inputs very closely. 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 9 microseconds wide for each of the high and low states, they should be correctly sensed. Pulse widths less than 8.7 microseconds will usually be incorrectly processed! The effective maximum stepping rate is therefore 17.4 microseconds per microstep (both motors may be stepped at the same time), thus providing for a maximum step rate of about 57,600 microsteps per second per motor.

If mode 1 is used, then each input line ('x-', 'x+', 'y-', 'y+') is independently monitored for pulse edges, and is used to request a single step in the indicated direction.

If mode 2 is used, then each input line pair is used to control step and direction. 'x-' and 'y-' are used to determine the direction the indicated motor will spin on an associated step request (low means spin minus, high means spin plus). The 'x+' and 'y+' inputs are monitored for the related step requests: a low-going edge on the indicated line generates a step request on the associated motor. The restriction of timing is that each direction line ('x-' or 'y-') must be stable at least 20 ns before the low-going edge of the associated step line ('x+' or 'y+'), and must remain stable for at least 8 microseconds.

If the extra feature of "Limit switch control of the motor current" is requested (for example through the mode of "6E"), then the limit switches are interpreted as follows:

Limit Switch	Description
LY-	High means enable the Y motor, low means disable the Y motor (that is to say, if LY- is low, the Y motor is off)
LY+	If the Y motor is enabled (LY- is high), then LY+ controls the motor current used. High means use full current, low means use ½ current.
LX-	High means enable the X motor, low means disable the X motor (that is to say, if LX- is low, the X motor is off)
LX+	If the X motor is enabled (LX- is high), then LX+ controls the motor current used. High means use full current, low means use ½ current.

The "Step-And-Seek" mode of operation (adding '8' to the command mode) changes the behavior from being direct instant steps generated on the motors to being relative seek requests, subject to the 'K', 'R', and 'P' commands. This allows a constant pulse rate to be used to request steps, while actually generating a trapezoidal ramp profile for the motor.

"Step-And-Seek" allows all of the normal operation of the firmware to continue to occur, except for the SLEW inputs being reprogrammed as step-and-direction inputs. The limits are treated as limits -- that is to say, the special step-and-direction mode of redefining limits as power levels is actively disabled.

Version 5.40 of the firmware made extensive modifications to this mode, in order to support incoming pulse rates that are less than the target rate.

When in step-and-seek mode, the firmware reduces the top step rate ("R") to 46,080 steps per second, due to the increased processing overhead required during the interrupt management of the firmware. Additionally, the ramp rate ("P") gets reduced to the maximum perfect power of two that is less than or equal to the actively requested rate (for example, "20000P" gets reduced to "16384P").

e – Write EEProm data

The ‘e’ command is used to rewrite the EEProm data on the board. As of version 5.23 of the firmware, 'e' is ignored if you fail to provide a parameter (i.e., 'e' alone results in the board ignoring the command, while '0e' causes the normal 'e' operation with a parameter of '0').

The data value passed is used to control what kind of write action is to be performed, and is range-based. The data may be written and erased at least 100,000 times.

Value Range	Use
0-65535	Write indicated value into the current user-data ‘eeprom’ write address, then increment that address for the following write. This allows access to 32 words (64 bytes) of user-programmable memory
65536-65567	Specify next write address for user data write (above)
123456789	Write control portion of EEProm with current state of system. This saves all savable parameters (such as ramp rate, run rate, microstep size, I/O configuration, baud rates and so on) for use on the next reset of the board.
987654321	Erase the complete EEProm, restoring the system to factory defaults on the next boot.
Other	Illegal – firmware generates a verbose error response

Writing User data

The firmware supports 32 16-bit words (64 bytes) of user programmable EEProm. After a reset, the next EEProm write address is set to the first word of user data (User data address 0); from then on, each successive ‘e’ command with a data value of 0 to 65535 will write to the current user data address, and will then increment that address for the next write. If the new address would exceed 31, it stays at location 31.

To read that data back, use the ‘?’ command with a value of 16384+user.address. The code will reset the user-data access location to the desired address and report the current contents. It will also leave the pointer at that location, so that the next user data write will overwrite the location.

Alternatively, you may specify the next write address by using an ‘e’ command with a data value of 65536+user.address. This simply resets the current address to the desired value.

For example: to read and then update user data location 3, you could issue the commands:

```
16387?
     The board could respond with “X,16387,10” and an ‘*’
25e
     The board would respond with “e,25” (which confirms the write)
16387?
     The board would respond with “X,16387,25”
```

To simply write a ‘5423’ to location 15, you could issue the commands:

```
65551e
     The board would respond with just an ‘*’
5423e
     The board would respond with ‘e,5423’
```

Note that the code is intelligent enough to not actually erase and rewrite the data if the new value matches the old value.

Warning! You must wait for the microprocessor to respond with the standard command completion character '*' before transmitting any more commands! The firmware automatically reduces the clock rate to about 7.4 MHz while updating the EEProm, which will cause loss of serial data if any data is sent to the board during this event. Also, motor

motion actions are temporarily suspended during any 'e' command, to avoid corruption of the EEProm.

Saving the current firmware settings

The current firmware settings may all be saved by issuing the command:

123456789e

This saves everything that is capable of being saved: this includes such items as the ramp rates, start/stop rates, A/D control values, and all other key I/O port information. Items that are not saved are transient data such as current I/O port contents and current motor locations.

The new settings are used the next time the board is powered on, and the next time that a board reset (either soft via the '!' command or hard via the RST input) is issued.

Firmware settings that are saved by "123456789e"

The settings that are saved by the save firmware settings command are:

- A – Automatic Full-power Full Step rate trigger
- b – set baud rate
- C, D and U – User programmable TTL output values
- E – Enable or disable Remote Direct Pulse Control
- H – Half power mode
- K – Stop OK rate for each motor
- O – Step mode for each motor
- P – Slope rate for each motor
- R – Run rate for each motor
- r – all rate scaling parameters for analog control of each motor
- T – Limit and slew mask bits and modes
- t – configure I/O port direction and use
- V – verbose communications control, hex mode report control
- W – winding mode when motor idle
- ! – Microstep size for both motors

Restoring firmware settings to Factory Default values

The firmware EEProm may be reset to factory defaults by two methods.

1. Issue the '987654321e' command.
2. Do an "EEPROM RESET" hardware strap startup as is described below.

Both methods erase the EEProm flash memory, including any data that you have saved using the above "user data" write commands. This sets all of the memory to have a value of 65535, which is used by the firmware to tell it that the memory is uninitialized.

The first method purely erases the memory. It does not otherwise affect the current state of the board – that is to say, your current baud rates, start/stop rates and so on will remain unchanged until you reset the board (at which point the factory default settings become active).

The second method ("EEPROM RESET") is your emergency recovery method to allow you to regain control of a board which has unknown settings and a saved baud rate that cannot be discovered using our standard tools. The technique is:

1. Power off the board
2. Disconnect everything from the board except for the power connection and the USB connection (for safety)
3. Use a standard 0.1" shunt to jumper the "PGD" and "PGC" pins together that are on the 5 pin header on the top of the board (See below)
4. Power the board on
5. Confirm that you can 'talk' to the board (it will be communicating at 9600 baud); this is most easily done by running the 'TestSerialPorts' application
6. Power the board off
7. Remove the shunt from PGD and PGC
8. Reconnect your motors and wires as needed

The board should now be restored to its factory-default settings. If you still cannot 'talk' to the board, please contact us for further instructions.

The top 2 pins are PGD and PGC
(which you need to short together
In order to perform the reset action)

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

For firmware versions 5.00 through 5.04, 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.

Firmware versions 5.05 and later support a more intelligent goto, that will adjust the target location without stopping the motor, if it can get away with that action.

For example,

X

1000G

Y

-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).

K –Set the "Stop oK" rate

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

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

For example,

100K

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

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

57600K

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

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

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

The latched events reported are as follows:

Bit	Value	Description
0	+1	Y- limit reached during a Y- step action
1	+2	Y+ limit reached during a Y+ step action
2	+4	X- limit reached during a X- step action
3	+8	X+ limit reached during a X+ step action
4	+16	System power-on or reset ("!") has occurred
5	+32	If set, the board is running using a precision clock reference (special order option)
6	+64	If set, the EEPROM RESET strap is installed. The board will not run motors until the RESET strap is removed!
7	+128	If set, all motor motion is blocked by the state of the NXT input (see the 't' command for configuration of this effect)

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
*
```

Bits 5 and 6 are 'sticky': that is to say, they will not go away until you repower or reset the board.

Bit 5 tells you that the board has an optional factory-installed precision clock reference that is being used for all timing (which makes the rates be accurate to about 100 ppm).

If bit 6 is set, then the board has detected that the EEPROM RESET strap is installed (a jumper between the PGC and PGD pins on the 5 pin header in the middle of the board). As long as this strap is installed, the board cannot run motors – it must be removed for normal operation.

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.

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

1M : 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 8 steps per complete full step, and performs a near-constant-torque calculation for positions between full step locations. The other modes include two full step modes and an alternating mode. For the full step modes, one enables only 1 winding at a time (low power), while the other enables 2 windings at a time (full power). The remaining mode alternates between 1 and 2 windings enabled.

The values which control this feature are:

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

For example,

0O

sets the above ½ power full step mode, while

3O

sets the default microstep mode.

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

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

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

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

The slope can be specified to be from 1 through > 1,600,000,000 microsteps per second per second. If a value of 0 is specified, the code forces it to have a value of 8000.

PLEASE NOTE: When the firmware is told to operate in its "Step-And-Seek" mode (via the 'E' command), the ramp will be forced to be the maximum power of 2 that is less than or equal to the ramp that you request. For example, "20000P" will be reset to "16384P", in order to make interrupt-time calculations feasible.

Firmware versions 5.48 and later support a separate ramp rate for limit events. This second rate is selected by using a negative ramp (such as -20000P) to override the standard ramp rate. Note that setting a positive ramp rate (such as 10000P) sets BOTH the standard ramp rate and the limit ramp rate to the selected value. You then separately request the limit ramp rate (using a negative value), which then replaces the limit ramp rate with your request.

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:

250P

500R

2000G

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 57,600 microsteps per second. If a value of 0 is specified, the code switches to use of the potentiometer as its rate source. 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 1-57,600 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 57,600 update tests per second. The motor windings are then updated according to the stepping mode. For example, if the stepping mode (the 'o' command) for a given motor is one of the full-step modes instead of the microstep mode, and the microstep resolution is set to '1', then the motor will actually experience motion at 1/64th of the specified rate.

For example,

X

250R

Y

1000R

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.

If you are currently executing a targeted GoTo or Slew command which has a specific target location (i.e., "2000G" or "-300S"), the new rate will not take effect until the motion has completed. If you are executing a generic "Slew in a given direction" command ("+S" or "-S"), the new rate will take effect immediately, and the motor will change its rate to match the request using the current "P" (ramp-rate) value.

Note that if you are operating in the "Step-And-Seek" mode of the firmware (see the "E" command), the maximum allowed value for the run rate is 46,080 microsteps per second.

r – Set rate scaling parameters for X, Y motor rate control

The ‘r’ command is used to scale A/D readings into useable rate values when the NXT and/or RDY connections are used as motor rate control inputs (see the ‘t’ command, t – Configure all I/O port directions and use).

By default, if the ‘r’ set of commands is not given for a given motor, the system is set to scale a ‘0’ A/D reading into a rate of ‘1’, and a 5 volt 4095 A/D reading into the maximum rate of 57,600. Through use of the microcoded ‘r’ commands, the voltage inputs may be fully scaled into real rates, with both minimum and maximum rates being specified as part of the configuration. Additionally, you may scale a high voltage (such as 5 volts) into the minimum desired rate, with a low voltage (like 1 volt) into the maximum rate that you want.

The 'r' command is actually a set of commands, microcoded by their low 3 bits. The 'r' command mostly operates on a set of temporary ("pending") registers, which then become active with the special 'Accept new pending data' command.

Use of 'r' consists of:

1. Reset the 'pending data' to default values, so you know your starting point
2. Issue the appropriate 'r' commands to change from those defaults to new values as needed
3. Issue the 'accept new pending data' command to activate the new scaling
4. To actually use the potentiometer for control, you then need to set the rate of the associated motor to 0 ("0R").

The encoding is as shown in the following table:

Bits	Description
0-2	Microcoded command
3	If set, multiply data portion of command by 16 (allows 16 bit numbers to work, as with the Parallax Basic Stamp product line)
4-31	Data for the associated command

The commands that are currently supported are:

Command	Description
0	Reset pending A/D data to default values
1	Accept new pending data: Code calculates the new rate and scale factors as needed. If the new minimum rate is >= the new maximum rate, then A/D rate control is disabled.
2	Set the pending minimum rate value (default of 1): this is the lowest rate that will be generated when in POT mode
3	Set the pending A/D value that is to be associated with that minimum rate (0 to 4095, default is 0)
4	Set the pending maximum rate value (default of 57,600): this is the maximum rate that will be generated when in POT mode
5	Set the pending A/D value that is to be associated with that maximum rate (0 to 4095, default is 4095)
6	<reserved> Do not use
7	<reserved> Do not use

Note that the system explicitly supports 'reverse slope' mappings: that is to say, you can associate a high A/D reading (such as 4000) with a low rate (10), while a low reading (50) is associated with a high rate (35000). Also, the rates generated by the code are clipped to the values that you specify in subcommands 2 and 4: this allows you to have full control over the actual rates generated by the system.

Please note that you CANNOT generate a rate of 0! That is to say, mapping an A/D reading into 0 with the intent of strictly using the potentiometer to stop the motor will not work. Code in the firmware that operates after the rate conversion code clips the real rate used to the legal limits of the firmware (1 to 57,600).

As is shown in the table, bit 3 may be used to multiply the data-part of the command by 16. This allows controllers which can only generate 16 but unsigned integers to still provide for rates that would otherwise generate too large of a number.

The easiest way to understand the command encoding is that you add the command (0 to 5) to 16 * the data value

As an example sequence, to set the mapping of:

0 volts → 100

3 volts → 2000

you would send the sequence:

0r	Reset all pending values to their defaults
1602r	Set the minimum rate to 100 (16*100 -> 1600)
	The A/D value of 0 is already set as a default value
32004r	Set the maximum rate to 3200 (16 * 2000 → 32000)
39317r	Set the max A/D value to 3 volts (3/5 * 4096 → 2457,
	2457 * 16 → 39312, 39312 + 5 → 39317)
1r	Accept the new data as the new scalings

You would then actually enable use of the above by setting a target rate of 0:

0R

Also note: the code supports reporting to you the above settings, via the '-16' through '-19' report commands:

-16?	Report A/D based (POT control) min rate value
-17?	Report A/D value at above min rate value
-18?	Report A/D based (POT control) max rate value
-19?	Report A/D value at above max rate value

You should use the above commands when testing your scaling parameters, to make certain that you have correctly encoded your request.

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 only "+" or "-" (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, while

-250S

will invoke the "relative seek" calculation mode of the firmware.

When doing a relative seek (i.e., "-250S"), the address calculations are normally based on the current TARGET location, not the current instantaneous location. The actual rules are as follows:

1. If the given motor is currently executing a GoTo or relative Seek command, then the new location is calculated as a delta from the old target. For example,

```
Current State:
    Our current location is 1000
    Our current target is 2000
    We are doing a GoTo action
Request:
    -500S
Calculation:
    Since we are doing a normal GoTo,
    the new target location will be "2000-500", or 1500
Result:
    Motor stops, then goes forward to location 1500
```

2. Otherwise, the current location is treated as the value to calculate from for the relative motion. For example,

```
Current State:
    Our current location is 1000
    We are executing a "+S" command (slew positive)
Request:
    -500S
Calculation:
    Since we are executing a Slew,
    the new target location will be "1000-500", or 500
Result:
    Motor stops, then goes backward to location 500
```

This was set up this way as being a reasonable compromise on the intent of the meaning of "relative". If you want to force the motion to be strictly relative to the current location, you issue the "Z" (stop) command first. Once that has been issued, the motor is placed in a special state (stopping, no target), which permits relative slew to be from the current location.

For example, to go -500 steps from the current location, regardless of whether the current action is a slew or a targeted goto, issue the command:

Z

-500S

T – limit, slew switch control and motor driver enable

The limi'T' switch command is used to control interpretation of the board limit and slew switch inputs, as well as control of the motor output drivers. **By default, 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.

As of version 5.23 of the firmware, 'T' is ignored if you fail to provide a parameter (i.e., 'T' alone results in the board ignoring the command, while '0T' causes the normal 'T' operation with a parameter of '0'). The command takes a bit-encoded parameter, which lists which switches are to be blocked from action. The values are:

Bit	Numeric Sum Value	Action
0	+1	Block LY-
1	+2	Block LY+
2	+4	Block LX-
3	+8	Block LX+
4	+16	Sense level, LY-
5	+32	Sense level, LY+
6	+64	Sense level, LX-
7	+128	Sense level, LX+
8	+256	Block slew Y-
9	+512	Block slew Y+
10	+1024	Block slew X-
11	+2048	Block slew X+
12	+4096	Block Y drivers from motor control, placing them into TTL Output control (for access via the 'C', 'D' and 'U' commands)
13	+8192	Block X drivers from motor control, placing them into TTL Output control (for access via the 'C', 'D' and 'U' commands)

Bits 8-11 are used to free up the slew inputs for use as generic TTL inputs. If they are disabled using this command, then they may be used as generic TTL sensors in your application.

Bits 12 and 13 control whether the motion control system actually makes it through to the motor driver outputs. If a motor driver is disabled using this command, then that set of 4 outputs becomes available for use with the TTL output commands "C", D" and "U" (providing for high-current outputs under software control).

The limit sense level bits are used to define the input level for the indicated limit input lines which are used to stop motor motion. A 0 means "use a logic low to stop", while a 1 means "use a logic high to stop". By default, the system uses a logic low to stop, so that the inputs (which are internally pulled high) will not cause a motor to stop if they are not connected.

For example,

4T

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

240T

would invert the sense of all of the limit input sensors, so that a low means "operate" and a high means "limit reached".

Note that the current input values for the limit and slew switches can be retrieved with the '5?' (limit switch) and '6?' (slews plus extras) commands, as described in the Other report values section of this manual.

t – Configure all I/O port directions and use

The 't' command is the master tool used to control the I/O port directions and firmware use of all of the programmable I/O ports in the BS1010 product. As of version 5.23 of the firmware, 't' is ignored if you fail to provide a parameter (i.e., 't' alone results in the board ignoring the command, while '0t' causes the normal 't' operation with a parameter of '0'). It is bit encoded into several groups, each of which control different ports and features.

Bits 0-2: RDY I/O configuration

These 3 bits define the use of the RDY signal

2 1 0	Value	Description
0 0 0	(+0)	TTL Output
0 0 1	(+1)	TTL Output, contains motion complete flag (0=busy, 1 = done)
0 1 0	(+2)	TTL Input
0 1 1	(+3)	TTL Input, extended feature (unimplemented: do not use at present)
1 0 0	(+4)	A/D input (generic use)
1 0 1	(+5)	A/D input, controls X rate
1 1 0	(+6)	A/D input, controls Y rate
1 1 1	(+7)	A/D input, controls both X and Y rates

Bits 3-5: NXT I/O configuration

These 3 bits define the use of the NXT signal

5 4 3	Value	Description
0 0 0	(+0)	TTL Output
0 0 1	(+8)	TTL Input, low=disable motors, high=enable normal motor operation. See bit 17 for options on level sense.
0 1 0	(+16)	TTL Input
0 1 1	(+24)	TTL Input, triggers "Next Rate" action
1 0 0	(+32)	A/D input (generic use)
1 0 1	(+40)	A/D input, controls X rate, overrides RDY rate control if also specified for the X motor
1 1 0	(+48)	A/D input, controls Y rate, overrides RDY rate control if also specified for the Y motor
1 1 1	(+56)	A/D input, controls both X and Y rates, overrides RDY rate control if also specified

Bits 6-7: AN2 I/O configuration

These 2 bits define the use of the AN2 signal

7 6	Value	Description
0 0	(+0)	TTL Output
0 1	(+64)	TTL Input
1 x	(+192)	A/D analog input (forces bit '6' to 1 internally)

Bits 8-11: SI2, SO2, IO2 I/O configuration

These 4 bits define the use of the SI2, SO2 and IO2 signals

Bit	Value	Signal	Description
8	(+256)	SO2	High = configure as TTL input or Serial I/O, low = configure as TTL output
9	(+512)	SI2	High = configure as TTL input or Serial I/O, low = configure as TTL output
10	(+1024)	IO2	High = configure as TTL input, low = configure as TTL output
11	(+2048)	TTL Serial	TTL or Serial I/O for SO2/SI2: High = use as TTL serial (forces bits 8 and 9 high), low = configure as generic TTL signals

Bit 12: reserved for future use

Bit 12 is currently not used and is reserved for future expansion. Leave 0 for now.

Bits 13-15: Motion control configuration: Instant Stop and Double Current Modes

These 3 bits define extended motion control features

Bit	Value	Description
13	(+8192)	1 = Enable "Instant stop", motor X
14	(+16384)	1 = Enable "Instant stop", motor Y
15	(+32768)	1 = Enable "Double Current Mode" (motor X output is sent to both the X and Y connectors

The "instant stop" features allow the given motor to stop instantly upon detection of a blocking limit event, as opposed to slowing down and stopping as is normally done.

The "Double Current" mode sends identical signals to both the X and Y motor connectors, allowing them to be wired in parallel. This allows the board to handle up to twice the current (assuming that it is properly cooled). **Warning! If you wire the motor connectors in parallel and fail to configure the controller for double current mode, you WILL quite literally blow up the board! This is not a warranted failure!**

Please refer to the Single motor, double current mode of operation section of this manual for further information on the "double current mode" feature.

Bit 16 (+65536): Configure 'instant step' mode of operation on new motion request

This feature was introduced in firmware version 5.46. It adds the capability of issuing the first step pulse on a new motion ('G' or 'S' command) effectively immediately, instead of after a pause that is equal to the initial step rate.

All prior firmware versions operate using the pause. This was to allow the enabling of motor current (i.e., the transition from 'idle' to 'active' state) to have enough time to stabilize on the motor drivers themselves. On some applications, this delay could cause issues, so this feature is now capable of being controlled by bit 16:

Bit	Value	Description
16	(+65536)	1 = Enable "Instant step"

If you enable this feature, there is a key caveat related to motor winding control. If you are leaving motor windings enabled ('W' command), there are no issues. Otherwise, if you are operating in 1/2 or full step mode, you may have problems with the motor doing its first step in the wrong direction, since the current will not have stabilized the motor position before the motion starts.

You will need to test your application using this feature to make certain that the initial motor motion is correct!

Bit 17 (+131072): Configure TTL input sense level for NXT-based instant stop

This feature was introduced in firmware version 5.47. It adds the capability of allowing the user to specify the sense level (high or low) that is used to trigger the "stop" action of the NXT-based instant stop feature, from bits 3-5 of this command.

0 means low is stop, 1 means high is stop.

Common 't' settings to match prior firmware options

The 't' command can be used to make the firmware closely match behaviors of earlier separate firmwares that were available on the BS0610 and BS0710 product lines.

Manufacturing shipment default: 4065t

Signal	Value	Configuration
RDY	+1	Outputs READY signal
NXT	+32	Generic analog input
AN2	+192	Generic analog input
IO2	+1024	TTL Input
SI2/SO2	+2816	SI2/SO2 configured for SerRoute

This is the value used as the system default on an EEProm reset action.

Original GenStepper: 4057t

Signal	Value	Configuration
RDY	+1	Outputs READY signal
NXT	+24	'Next Rate' TTL input
AN2	+192	Generic analog input
IO2	+1024	TTL Input
SI2/SO2	+2816	SI2/SO2 configured for SerRoute

GenStepper configured for double current mode: 36825t

Signal	Value	Configuration
RDY	+1	Outputs READY signal
NXT	+24	'Next Rate' TTL input
AN2	+192	Generic analog input
IO2	+1024	TTL Input
SI2/SO2	+2816	SI2/SO2 configured for SerRoute
X/Y	+32768	Configure X and Y connectors for double current mode

PLUS: Block Limits LY+, LY- and slew Y+/Y- via the '771T' command.

PotStepper: 4089t

Signal	Value	Configuration
RDY	+1	Outputs READY signal
NXT	+56	Configure as analog rate for X and Y motors
AN2	+192	Generic analog input
IO2	+1024	TTL Input
SI2/SO2	+2816	SI2/SO2 configured for SerRoute

PotStepper, dual potentiometer mode: 4085t

Signal	Value	Configuration
RDY	+5	Analog rate for X motor
NXT	+48	Analog rate for Y motor
AN2	+192	Generic analog input
IO2	+1024	TTL Input
SI2/SO2	+2816	SI2/SO2 configured for SerRoute

U – Set TTL Lines low that are '1' in 'x'

'U' is used to set selected bits low on any of the TTL ports that are enabled as TTL outputs. It is bit encoded as:

Bit	Hex Value	Decimal Value	Description
0	0x0001	1	Y WA1, if Y drivers enabled (see bit 9)
1	0x0002	2	Y WA2, if Y drivers enabled (see bit 9)
2	0x0004	4	Y WB1, if Y drivers enabled (see bit 9)
3	0x0008	8	Y WB2, if Y drivers enabled (see bit 9)
4	0x0010	16	X WA1, if X drivers enabled (see bit 8)
5	0x0020	32	X WA2, if X drivers enabled (see bit 8)
6	0x0040	64	X WB1, if X drivers enabled (see bit 8)
7	0x0080	128	X WB2, if X drivers enabled (see bit 8)
8	0x0100	256	Enable X driver outputs, if enabled for this command via the 'T' command
9	0x0200	512	Enable Y driver outputs, if enabled for this command via the 'T' command
10	0x0400	1024	RDY
11	0x0800	2048	NXT
12	0x1000	4096	AN2
13	0x2000	8192	SO2
14	0x4000	16384	SI2
15	0x8000	32768	IO2

Add the decimal value for each bit that you want to set to '0' (low), and issue the command on the sum. Those bits will get cleared: the others will be unchanged.

Note that you need to use the 'T' command (T – limit, slew switch control and motor driver enable) in order to enable access to the motor output drivers, and the 't' command in order to enable access to the other TTL I/O lines. Note also that if you are using the motor driver outputs (from the 'X' or 'Y' connectors), you must have the given output enabled in order for it to actually drive the output high or low (see bits 8 and 9).

For example, to set the IO2 line low, you could issue the command:

32768U

(assuming that IO2 was configured as a TTL output).

There are two other commands which may be used to control the output port levels:

- C – Set TTL lines high that are '1' in 'x'
- D – Set the TTL lines to the requested values

V – Verbose mode command synchronization and processor clock control

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. As of version 5.20 and later of the firmware, it also includes the capability of resetting the processor clock rate. **By default (after power on and after any reset action), the board is normally 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. It is also configured to operate the processor clock at full speed (about 29.5 MHz).** 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. As of version 5.23 of the firmware, 'V' is ignored if you fail to provide a parameter (i.e., 'V' alone results in the board ignoring the command, while '0V' causes the normal 'V' operation with a parameter of '0').

The firmware actually recognizes and responds each new command within a few microseconds of receipt of the stop bit of the received character. 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 slower non-interrupt based devices can change their state! In microprocessor, non-buffering designs (such as with the Parallax, Inc.[tm] Basic Stamp [tm] series of boards), this can be a significant issue. The firmware handles this via a configurable option in the 'V' command. If enabled, the code will delay 1 millisecond upon receipt of a new command character. This really means that the first data bit of a response to a command will not occur until at least 9 bit intervals after completion of transmission of the stop bit of that command (about 900 uSeconds at 9600 baud); for the Basic Stamp[tm] this is quite sufficient for it to switch from send mode to receive mode.

The 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 millisecond before transmission of first character of any command response.
2	+4	Send numeric responses in hexadecimal instead of decimal.
3	+8	Used in conjuction with Bit 4 to reset the processor clock speed. If Bit 4 is set, then bit 3 defines the clock rate: +0: Use standard high speed clock (29.5 MHz) +8: Use low speed clock (7.4 MHz)
4	+16	If set, then use bit 3 to redefine the processor clock rate. If not set, then the clock rate is left unchanged, and bit 3 is ignored.
5	+32	If set, code always sends responses: incoming characters do NOT stop pending data. WARNING -- This can cause loss of incoming commands, if 4 or more characters sent while the 64 character transmit buffer is full!

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

For example,

0V

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

3V

would enable transmission of the <CR><LF>sequence, preceded by a 1-character delay.

The complete table of options when not changing the processor clock rate is:

Value	Delay First	<CR><LF>	Hex mode reports
0	No	No	No
1	No	Yes	No
2	Yes	No	No
3	Yes	Yes	No
4	No	No	Yes
5	No	Yes	Yes
6	Yes	No	Yes
7	Yes	Yes	Yes

Changing the processor clock rate using the 'V' command

The 'V' command may also be used to change the processor clock rate for the board. The firmware currently supports two rates: 29.5 MHz (the default) and 7.4 MHz (low power). When the board is fully idle, with the motor drivers disabled and no TTL lines configured as outputs, then operating at the reduced clock rate changes the board draw from its standard 150 mA down to 90 mA. This is quite useful to keep the processor temperature down, and to reduce current draw during idle time on battery applications.

The side effects of changing the processor clock rates to the lower clock speed can be summarized as:

1. All rates (i.e., 'K', 'R' and 'P') have a new reduced top limit of 14,400 steps/second, as opposed to the 57,600 standard value.
2. If operating using the Step And Direction mode of action, the minimum pulse width becomes 36 microseconds, instead of 9.
3. If you currently have any of the standard rates set to a value above 14,400, you will need to re-issue the matching command to reset that rate to a legal value. Otherwise, the board response will be erratic.
4. Motors are likely to 'sing' more, since the update rate for the windings will be four times slower.

Warning! It is an absolute requirement that your code wait for the '*' response whenever a 'V' command is given that resets the clock rate! The firmware has to reset the baud rate of the serial clock to match the new processor rate. This action will garble any pending incoming data, causing either incorrect command execution (since the data may become almost anything), or missed command execution.

To change the clock rate, you add one of two values to your standard 'V' command (from the above table):

+16: Set to the normal high speed clock (29.5 MHz)

+24: Set to the low speed clock (7.4 MHz)

For example, to set to the low speed clock with the common <CR><LF> responses, issue the command:

25V

To set the high speed clock with reduced response and the 1 millisecond delay, send:

18V

Remember to wait for the '*' before you send any more serial data to the board!

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,

X

100R

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

Y – Select motor Y

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

For example,

Y

100R

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, 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,

X

Z

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.

As of firmware version 5.23, if just '!' is provided without any parameter, the effect is identical to that of issuing the '0!' command, which resets the board to its power-on EEProm-requested settings.

The value passed is used as the new microstep size, in fixed 1/64th 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 1/16th 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, 32 and 64 (giving you true microstep step sizes of 1/64, 1/32, 1/16, 1/8, ¼, ½ and 1 respectively). All other values (such as RATE or GOTO LOCATION) are then expressed in units of the microstep size; therefore, location "3" would mean "3/64" in the finest resolution (microstep set to 1), and "3" in the largest resolution (microstep set to 64). Note that the ability to specify 64 started with version 1.75; all earlier versions had an upper limit of 32/64th of a step (1/2 step) as the largest step size.

For example,

4!

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

There are 2 special values reserved for the '!' reset command:

0! – Reboot using the EEProm-specified settings, including the microstep size

-1! – Reboot using the factory default settings

The reset command also resets all other settings to their EEProm-specified values: By default, these are:

- **4!** – Set the microstep size to 1/16 (4/64ths) of a step per logical step.
- **3072A** – Set the Automatic Full Step rate to be >=3072 microsteps/second
- **B** – Select both motors for the following actions
- **9600b** – Set the communications baud rate to 9600 baud
- **0E** – Operate in normal (non-step-and-direction) mode
- **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** – (three-oh) 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 motor to 800 microsteps/second
- **'r' settings** – Set the POT rate scales such that 0 volts maps into a rate of 1, and 5 volts maps into a rate of 57,600 for both motors
- **0T** – Enable all limit switch detection
- **4065t** – Set all I/O ports to our defined default settings

- **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,

X

2000=

Y

4000=

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.

X

2000=

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:

Negative values: special value reports

Values 5 and 6: I/O port reports

Other positive value: microcoded memory locations

The most commonly used reports are:

0: Report items -1 through -11 of the special reports

-1 to -19: 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
- -13; Report I/O configuration ('t' command)
- -14; Report microstep size
- -15; Report maximum step rate for this firmware
- -16; report A/D based (POT control) minimum rate value
- -17; report A/D value at above minimum rate
- -18; report A/D based (POT control) maximum rate value
- -19; report A/D value at above maximum rate
- 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 (or a semicolon if HEX MODE reporting is enabled from the 'V' command).
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. An "*" 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 reportable items -1 through -11

This 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.

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,

```
-12?
```

could report:

```
*
GenStepper 5.00 Nov 4, 2009
Copyright 2003-2009 by Peter Norberg Consulting, Inc. All Rights
Reserved.
*
```

-13: Report I/O configuration ('t' command data)

This reports the I/O configuration data as used by the "t" command (t – Configure all I/O port directions and use).

For example,

```
X-13?
```

Would report the current I/O configuration. You could receive:

```
*
X,-13,4095
*
```

-14: Report microstep size

This reports the current microstep size (parameter for the '!' command)

For example,

```
X-14?
```

Would report the current microstep. You could receive:

```
*
X,-14,4
*
```

-15: Report the maximum step rate supported by this firmware

This reports the maximum step (and ramp) rate (in microsteps/second) supported by this firmware release.

For example,

```
X-15?
```

Would report the maximum rate. You could receive:

```
*
X,-13,57600
*
```

-16: Report A/D based (POT control) minimum rate value

This reports the minimum speed which may be attained using the A/D rate mode of control for this motor.

For example,

```
B-16?
```

Would report the minimum POT rates on both motors. You could receive:

```
*
X,-16,1
Y,-16,20
*
```

-17: Report A/D setting at minimum rate

This reports the A/D setting at the minimum speed which may be attained using the A/D rate mode of control for this motor.

For example,

```
B-17?
```

Would report the A/D settings at the minimum POT rates on both motors. You could receive:

```
*
X,-17,0
Y,-17,4095
*
```

-18: Report A/D based (POT control) maximum rate value

This reports the maximum speed which may be attained using the A/D rate mode of control for this motor.

For example,

```
B-18?
```

Would report the maximum POT rates on both motors. You could receive:

```
*
X,-18,13500
Y,-18,23000
*
```

-19: Report A/D setting at maximum rate

This reports the A/D setting at the minimum speed which may be attained using the A/D rate mode of control for this motor.

For example,

```
B-18?
```

Would report the A/D settings at the maximum POT rates on both motors. You could receive:

```
*
X,-17,4095
Y,-17,0
*
```

Other report values

5: Report LIMIT inputs

This is used to provide a snapshot of the LIMIT SWITCH input values. The report value is bit mapped as follows:

Bit	Value	Description
0	+1	LY- (Y- limit)
1	+2	LY+ (Y+ limit)
2	+4	LX- (X- limit)
3	+8	LX+ (X+ limit)

6: Report SLEW and IO inputs

This is used to provide a snapshot of the SLEW and IO port input values. The report value is bit mapped as follows:

Bit	Value	Description
0	+1	Y- (Y- slew)
1	+2	Y+ (Y+ slew)
2	+4	X- (X- slew)
3	+8	X+ (X- slew)
4	+16	NXT, when configured as TTL input
5	+32	RDY, when configured as TTL input
6	+64	SI2, when configured as TTL input
7	+128	SO2, when configured as TTL input
8	+256	IO2, when configured as TTL input
9	+512	AN2, when configured as TTL input

Note: The RDY line (bit 5, +32) is automatically switched to reporting the logical 'Ready' status of the board (neither motor is moving) if RDY is configured as analog or as TTL output.

16384 to 16415: Report User EEProm data

This report echoes back the contents of the user data within the EEProm. As is described in the documentation for the 'e' command, there are 32 user-data words available. A specific side effect of this report is to set the next EEProm write address to refer to the data word that was just reported.

User data word 0 is mapped into report value 16384 (i.e., add 16384 to the user data element ID to determine the report value to use to extract the data).

16416 to 16447: Report fixed firmware configuration data

This report provides information from the fixed configuration region of the board.

Value	ID	Description
16416	0	Board type
16417	1	Board serial number within manufacturing run
16418	2	Manufacturing run number for board
16419	3	Manufacturing reference number for board
16420	4	Special order firmware flags
16421-16447	5-31	Reserved for future expansion

Board type: This describes the type of board. At present, 0 is defined as the BS1010/SS1010 product.

Board serial number within run: unique serial number within manufacturing run for this board.

Manufacturing run number: Board run encoded information. This number, combined with the board serial number, provides a unique board identifier. The manufacturing run number is encoded as:

Bits 0-15: Raw run number

Bits 16-23: Manufacturing process code.
If this 8-bit code may be represented as an upper case letter (i.e., if it has numeric value of 65 through 90), then it is reported that way. Otherwise, it is reported as a 2-digit hexadecimal value.

The serial number/manufacturing run number pair are usually reported as:

LRRR-SSS

Where L is the process code, RRR is the run number, and SSS is the serial number.

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.

Additional notes on Direct TTL Step Control

The '1E' command (see the 'E' command under 'Serial Control' for complete documentation) 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 57,600 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 NORMALLY IGNORED DURING THIS MODE OF OPERATION, UNLESS SPECIAL FIRMWARE OPTIONS ARE ORDERED OR SPECIAL CURRENT CONTROL REQUESTS ARE MADE.**

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 8 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 normally 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.

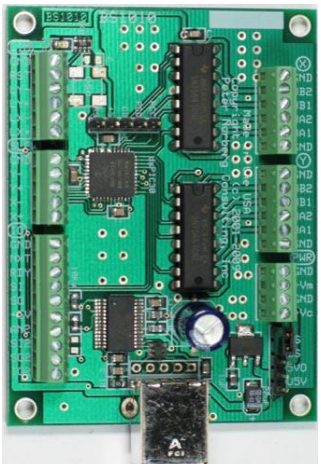
All of the other commands may be used with no negative effects on timing in the system.

StepperBoard.dll – An ActiveX controller for StepperBoard products

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

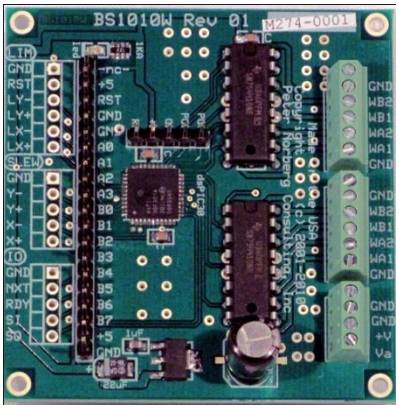
The program is well-documented in the manual StepperBoardClass.pdf. Please refer to that manual for more information about the product.

Board Connections



AR-BS1010-TTT

AR-BS1010-LLL



AR-BS1010W-PTT

AR-SS1010-LLL

Board Size

The AR-BS1010 boards, oriented as shown on this page, are 3.0 inches high by 2.25 inches wide.

The AR-BS1010W boards, oriented as shown on this page, are 2.5 inches high by 2.4 inches wide.

The AR-SS1010 boards, oriented as shown on this page, are 3.0 inches high by 1.75 inches wide.

Mounting Requirements-BS1010

The boards may be mounted using four #4 or #5 machine screws. The holes are 0.125 inches in diameter, and are positioned exactly 0.125 inches in from each corner. They allow up to a number 5 screw, which thus allows use of the standard #4 mounting spacers. Vertically, their centers are 2.75 inches apart, and horizontally they are 2.00 inches apart. Thus, when the board is positioned as shown above, their positions are:

(0.125, 0.125), (2.125, 0.125),
(0.125, 2.875), (2.125, 2.875)

Mounting Requirements-BS1010W

The boards may be mounted using four #2 machine screws. The holes are 0.1 inches in diameter, and are positioned exactly 0.1 inches in from each corner. Vertically, their centers are 2.75 inches apart, and horizontally they are 2.00 inches apart. Thus, when the board is positioned as shown above, their positions are:

(0.1, 0.1), (2.3, 0.1),
(0.1, 2.4), (2.3, 2.4)

Mounting Requirements-SS1010

The boards may be mounted using four #4 or #5 machine screws. The holes are 0.125 inches in diameter, and are positioned exactly 0.125 inches in from each corner. They allow up to a number 5 screw, which thus allows use of the standard #4 mounting spacers. Vertically, their centers are 2.75 inches apart, and horizontally they are 2.00 inches apart. Thus, when the board is positioned as shown above, their positions are:

(0.125, 0.125), (1.625, 0.125),
(0.125, 2.875), (1.625, 2.875)

Connector Signal Pinouts, BS1010 and SS1010

There are eight connectors on each board.

Going from top-left down, we have:

- Debugger connector (5 pin SIP header)
- TTL And Analog I/O 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)), AN2, SI2, SO2, IO2.

Then, on the bottom we have:

- a USB female connector on the bottom of the board.

Finally, on the BS1010, 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)

The SS1010 differs only in the order of connectors on the right side:

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

Connector Signal Pinouts, BS1010W

There are eight connectors on each board.

Going from top-left down, we have:

- Debugger connector (5 pin SIP header)
- TTL And Analog I/O signals, matching 'new' usage :
  - 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))
- TTL And Analog I/O signals, combined on one 19 pin SIP header

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

- X Motor connector (upper right)
- Y Motor connector (center)
- Power connector (lower right: motor and logic power are from the same pins)

Debugger connector

Pin (numbered left-to-right)	Name
1	RST
2	+5V
3	GND
4	PGD
5	PGC

Firmware Factory Reset – Short PGD and PGC together

Pins 4 and 5 (PGD and PGC) have a special extra function which may be used if you have set options in the board which make it unresponsive (such as setting the baud rate to a value that you have forgotten). If you short those two pins together (using a standard shunt), the firmware will reset itself to its default values (including the standard 9600 baud communications setting). Motor control is disabled as long as those pins are shorted; so your process for performing a firmware reset is:

1. Turn off power to the board
2. Disconnect all wires from the board except for power
3. Short PGC and PGD together
4. Apply power to the board for at least 10 seconds (if the board is USB powered, wait until the 'LED' turns on).
5. Turn off power
6. Remove the PGC-PDG short
7. Reconnect your board as usual: it should now be at its factory default settings.

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 connector is used to warn the firmware 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 pulled up to +5 with 1K resistors.

Note that you may use the 'T' command (T – limit, slew switch control and motor driver enable) to change the response of the board to the limit inputs.

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. By default, they each cause the indicated motor to turn at its current rate in the indicated direction, as long as the indicated signal is LOW. As with the limit inputs, 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. Y- through X+ are pulled up to +5 with 1K resistors

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.

Note that you may use the 'T' command (T – limit, slew switch control and motor driver enable) to change the response of the board to the slew inputs.

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 Serial Input (TTL level)
SO	OUTPUT: Raw Serial Output (TTL Level)
+5V	Board +5V, may be used as tie point for potentiometer rate control
AN2	Normally set up as an analog input
SI2	Daisy-chain SerRoute-oriented TTL serial input
SO2	Daisy-chain SerRoute-oriented TTL serial output
IO2	Generic TTL I/O port

Note that the +5 through IO2 signals are not available on the BS1010W

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

NXT is normally configured as an Analog input (channel 0, as is available through use of the 'a' command). It can be reconfigured to be used as a generic TTL I/O signal, as a "next rate" TTL input (compatible with prior versions of GenStepper), and as an analog input (0 to 5 volt) used for rate control of either or both motors. Use the 't' command to reconfigure this port for any of these uses.

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. As with the NXT signal, RDY may be reconfigured as a generic TTL I/O signal, an analog input (0 to 5 volt), and as an analog input (0 to 5 volt) used for rate control of either or both motors. Use the 't' command to reconfigure this port for any of these uses.

SI and SO are the "real" serial input and serial output (respectively), as seen by the on-board microprocessor chip. If the application desires direct TTL serial communications (for example, if the Parallax Inc.[tm] Basic Stamp [tm] based products are being used to control the board), use these pins. If the USB connection is also present, it will be ignored for serial input to the board.

AN2 is a generic I/O pin, normally configured as an analog input (0 to 5 volt). It may also be reconfigured as a general TTL output or TTL input through use of the 't' command.

SI2 and SO2 are normally configured as SerRoute-style daisy-chained serial communications, allowing multiple boards to be operated off of one USB port. They may also be reprogrammed as generic TTL I/O pins though use of the 't' command. When configured as TTL-Serial, SI2 is the extended TTL-Serial input to the board, and SO2 is the extended serial output from the board.

IO2 is a generic TTL I/O signal, normally configured as a TTL input. It may be reconfigured as a TTL output through use of the 't' command.

BS1010W 19 Pin SIP Header

The BS1010W may have a 19 pin SIP header installed instead of the 3 LIM, SLEW and IO connectors. It combines all of those signals into one connector, for compatibility with the retired BS0506 product. The pins map into the LIM, SLEW and IO pins as:

SIP Name	New Name	Description
-nc-		unused pin
+5	+5	Access to +5 board power
RST	LIM: RST	Reset the microcontroller, when low
GND	LIM: GND	Ground reference for inputs
GND	GND	
A0	LIM: LY-	Y Minimum limit reached, when low
A1	LIM: LY+	Y Maximum limit reached, when low
A2	LIM: LX-	X Minimum limit reached, when low
A3	LIM: LX+	X Maximum limit reached, when low
B0	SLEW: Y-	Slew Y Negative
B1	SLEW: Y+	Slew Y Positive
B2	SLEW: X-	Slew X Negative
B3	SLEW: X+	Slew X Positive
B4	IO: NXT	Go to "next" step rate
B5	IO: RDY	Ready/busy output
B6	IO: SI	INPUT: Raw Serial Input (TTL level)
B7	IO: SO	OUTPUT: Raw Serial Output (TTL Level)
+5	IO: +5V	Board +5V
GND	IO: GND	

USB-B Serial (AR-BS1010 and AR-SS1010)

On the AR-BS1010 and AR-SS1010, there is a standard USB-B female connector for use with a standard USB-A-B cable to the computer.

BS1010 and SS1010 Power Connector (labeled here top-to-bottom) And Motor Voltages

Name	Description
GND	Ground for Vm
+Vm	4.5-vMax volts, for the X and Y motors vMax = 34 volts for the BS1010 series of boards, and 26 volts for the SS1010 series of boards.
GND	Ground for Vc
+Vc	Normally configured as +6.5-15 volts for the logic circuits. This may also be configured as +5.0 volts, bypassing the onboard regulator

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 and on the board version. 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 2940 voltage regulator.

In general, if the motors require a drive voltage between 6.5 and 15 volts, you are better off using a single power supply to operate the board. In this case, you would wire the power supply to the GND/+Vm power pair, and you would select the 'SS' setting for the power options jumper (as described on the following page).

If the motors require more than 15 volts to operate, **using the same power supply to the 2940 will cause the 2940 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 6.5 to 15 volts at 300 mA to the 2940 (the lower voltage you use, the better it is from a heat point of view). Use pins 1 (GND) and 2 (+Vm) to provide the motor power in this case, and ***SELECT THE CORRECT POWER SUPPLY JUMPER OPTION DEPENDING ON YOUR BOARD VERSION. OTHERWISE YOU WILL EITHER BE SHORTING THE POWER SUPPLIES TOGETHER OR SUPPLYING THE WRONG VOLTAGES TO THE LOGIC CIRCUITS***! **This jumper is described on the next page of this manual**.

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 6.5 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 5.5 to 7 volts, assuming that you do not want to exceed the 5 volt specification for the motor.

BS1010W Power Connector (labeled here top-to-bottom) And Motor Voltages

Name	Description
GND	Ground
GND	Ground (same as above)
+V	6.5 to 15 volt power for board
Va	6.5 to 15 volt power for board (same as above)

The power connector has two sets of power and ground pins. This is strictly for compatibility with the BS0506 approach to power: you just connect your power and ground to one of the above power (+V or Va) and ground (GND) pins.

The board only supports motors with voltage requirements in the range of 6.5 to 15 volts, since the same power supply is used to power both the on-board regulator and the motors.

BS1010 and SS1010 Power Option Jumper Configuration

The BS1010 and SS1010 boards (NOT the BS1010W) have a 4 or 5 pin SIP header positioned beside the power connector which controls how power is routed to the logic part of the board. The jumper must be installed in one of the 4 positions shown, **SS**, **DS**, **5VO** or **USB**. vMax is 34 volts for the BS1010, and 26 volts for the SS1010.

Option	Allowed Motor Supply Voltages	Use sep. Power supply	Comments
SS	6.5-15V	NO	Single power supply, connected to pins GND and Vm of power connector. Note that connecting power to the Vc will not work – it is not connected when this jumper is in the SS position.
DS	4.5-vMax	YES	Use two power supplies, one for the motor (connected to GND and Vm), the other for the digital power (GND and Vc). The digital power should be 6.5 to 15 volts, at least 300 ma. Note that lower voltages for the digital power are preferred, since they will result in less waste heat generated by the 2940 power regulator.
5VO	4.5-vMax	YES	Use two power supplies, one for the motor (connected to GND and Vm), the other for the direct digital power (GND and Vc). The digital power must be exactly 5 volts, at least 300 ma (the on-board regulator is bypassed). **Note that if the voltage exceeds 6 volts, the board will be permanently damaged. If it ever drops to 4.2 volts or less, the firmware will reset and lose all settings**.
USB	4.5-vMax	YES	If this option is present on your board, then you may power the board logic off of the USB cable power. The motors will still need to be powered from the Vm and GND power input pins, however.

Board mounting and cooling considerations

Note that if the current requirements are over about 0.6 Amp/winding on the BS1010 or 0.2 amp/winding on the SS1010, if the motor supply voltage is above 15 volts, or if the windings are left on through use of the 'W' command, 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! You should also use fan-based cooling if you are going to be operating the board in a warm environment (>100 degrees F), or if the board is running "hotter" than you like.

The board itself acts as the heat sink for the motor driver chips: most of the heat dissipation is from the **bottom** of the board, so having adequate air flow and clearance is a requirement. For low current operation (less than 150 mA and 12 volts), there are usually few issues; otherwise, the minimum mounting clearance needed is 1/4 inch for medium currents (150 mA to 600 mA or cases where the windings are left on at reduced levels continuously), and 1/2 inch for larger draws or for any other continuous duty operations.

We strongly suggest that you use fan-based cooling if the current requirements exceed 0.5 amps per winding.

Fan based cooling should be done such that both the bottom and top of the board in the area of the SN754410 or ULN2803 components are 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; however, a top-positioned fan which blows directly down on the SN754410/ULN2803 components at the 8-10 CFM level will also be fine as long as there is 1/2 inch or more of clearance under the board. **Do not use a fan which blows air away from the board**; this is completely ineffective in terms of cooling the system.

Calculating Current And Voltage Power Supply Requirements

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

1. Determine the individual motor winding current requirements.

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

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

V=IR

or, rearranging terms in order to find I,

I=V/R

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

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

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

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

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

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

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

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

3. Determine the voltage for your motor power supply

From the 'V=IR' formula (step 1, above), you can also derive the voltageyou're your system.

That is to say, the voltage (V) used to drive the motors should be that calculated from multiplying the desired single-winding current (I) by the resistance (R) of your motor windings.

For example, if you have a 12 ohm resistance in your windings, and you need to operate it at 0.5 amps of current, then the motor voltage will be 0.5 * 12, or 6 volts.

4. Determine the logic supply requirements

The logic supply normally requires 300 mA for the BS1010, SS1010 and BS1010W products. If you choose to operate a fan off of the on-board 5 volts in order to cool the system, or if you are going to tap the 5 volts for other external logic, you need to add in the current which is needed to operate the fan and the external logic. Do not exceed 500 mA! **Note also that if you draw more than 100mA from the board's 5 volt supply, you must fan-cool the board**.

The logic supply (+Vc) must always be in the range of 6.5 to 15 volts. If less than 6.5 volts is used, the regulator will not operate reliably (causing the board to reset itself, losing motor control and position information). If greater than 15 volts is used, you are likely to 'blow out' the logic voltage regulator, damaging the board.

5. Determine the power supplies you will be using

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

The logic supply must be in the voltage range of 6.5 through 15 volts. If your motor voltage requirements are outside of this range, then you will have to use a split supply (as described below) (or exactly 5.0 volts, if the '5VO' jumper is installed).

Single Supply.

If your motor power supply voltage is from 6.5 to 15 volts, then you may choose to use a single supply to operate the system. The current capabilities of the supply must exceed the sum of the current requirements of the motor(s) and the logic circuits.

Dual Supply

You may separate the motor supply from the logic supply. If you do so, we suggest using the lowest voltage in the range of 6.5 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 4.5 volts in all cases (due to some signal requirements on the board), and otherwise is as calculated under sections 1 through 3, above. If the supply is to drive 2 motors, please remember to double the current needs.

Note that when operated in dual supply mode, you may either operate your logic (Vc) voltage through our on-board voltage regulator, or you may provide exactly 5 volts of regulated DC to our board. Use of our on-board regulator is strongly encouraged!

The jumper options are "DS" for dual supply, using our on-board regulator, or "5VO" for dual supply, bypassing our regulator.

DS: You provide 6.5 to 15 volts DC to the +Vc input

5VO: You provide 5.0 volts regulated DC to the +Vc input

Note that in the 5VO position, **you will damage the board if your supply ever reaches or exceeds 6 volts**, and the board will undergo a reset if the voltage ever drops (even as a spike) to 4.2 volts.

Board Jumpers

The BS1010 and SS1010 boards have one jumper, used to configure power input to the board.

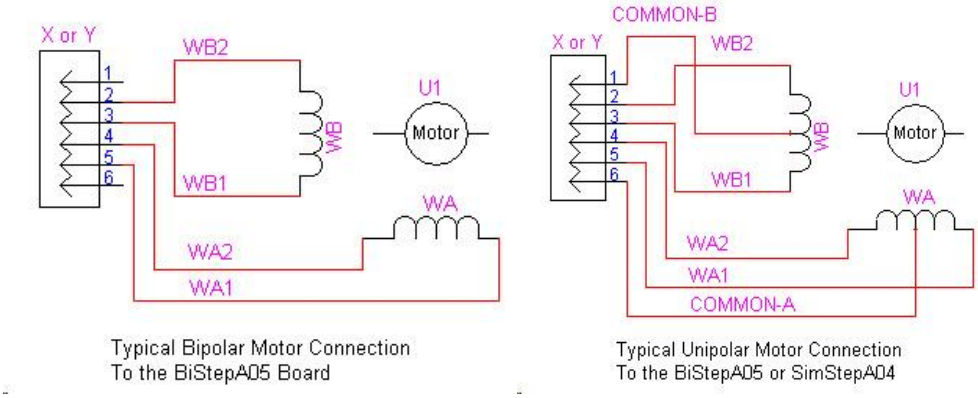
Power Selection Jumper - SS/DS/5VO/USB

This jumper is used to select how the board is to be powered. It may be factory hard-wired (for volume orders), or it will be available as a jumper selection as needed. The options are:

SS	Single-Supply operation. You must provide 6.5 to 15 volts at the Vm/Gnd input. Do not wire to Vc.
DS	Dual-Supply, regulator enabled operation. You provide 4.5 to 34 volts at Vm/Gnd (26 volt max for the SS1010) for the motor power, and 6.5 to 15 volts at the Vc/Gnd inputs.
5VO	Dual-Supply, regulator disabled operation. You provide 4.5 to vMax volts at Vm/Gnd for the motor power, and 5.0 volts at the Vc/Gnd inputs. vMax = 34 volts for the BS1010, and 26 volts for the SS1010 products. Note that if your 5 Volts ever drops to 4.2 volts or less, the board will act as if a RESET has occurred. Similarly**, if your supply goes to 6 volts or above, you will damage the components**.
USB	If this jumper position has a pin installed, then your board has the option of having the logic powered off of the USB 5 volt system by positioning the jumper here. You provide 4.5 to 34 volts at Vm/Gnd for the motor power.

Wiring Your Motor

There are two identical connectors used to operate the X and Y motors. The connectors are labeled with respect to which motor they operate. (This designation affects only which commands are to be used to control the motors; no other functionality is changed.) They are wired as follows for the BS1010 and the SS1010 series of controllers (pins counting from top to bottom):



Typical Bipolar Motor Connection
To the BiStepA05 Board

Typical Unipolar Motor Connection
To the BiStepA05 or SimStepA04

Pin	Name	Description
1	GND	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

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

On the SS1010 product, only unipolar motors are supported (as shown on the right-hand drawing). Note that pins 1 and 6 of the motor driver connector are internally connected to +Vm instead of ground on the SS1010 board!

Stepping sequence, testing your connection

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

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

Note also that it is explicitly legal when using the GenStepper firmware to operate your motor in "double current, ½ power mode". In "double current" mode, you wire your motor to both the X and Y motor connectors, and you jumper the board as described in the 'Configuring Double Current' mode section of this manual. In all other respects, you follow the rest of the instructions in this manual.

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

Determining Lead Winding Wire Pairs

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

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

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

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

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

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

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

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

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

Sequence Testing

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

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

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

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

Note that in the following discussions, a string of characters is shown as being typed by you (for example, `1000G`). Any of our boards will normally send a newline character each time they see a letter (such as the 'G' in the above sequence), and they will echo an asterisk (`*`) when they have completed processing the command. Therefore, your commands will all get an immediate response from the controller; **always wait for the '*' before you continue typing.**

Also, **all commands should be treated as case sensitive** (i.e., 't' may mean something different from 'T'). Please use the case of the command as shown in the following list.

For each pattern, request a motor motion in each direction using the sequence:

- Send: X
- Receive: * (i.e., wait for board to echo '*' back to you)
- Send: 800R
- Receive: *
- Send: 1000G
- Receive: *
- Send: I
- Receive: * (this may take a while; it will be sent by the board in response to the above 'I' when the motor stops)
- Send: 0G
- Receive: *
- Send: I
- Receive: *

- which should cause the motor to spin to logical location 1000 (at 800 microsteps per second), then return to location 0. Wait for the `*` response after each sub-command (the "`X`", "`R`", "`G`", and "`I`" commands) before typing the next character, in order to let the firmware finish processing the request.
- If you are not connected to the board in such a manner, you may test the board by first momentarily grounding the `X-`, and then the `X+` slew signals. While the signal is grounded, the X motor will try to spin in the associated direction.

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

A B C D, rewire as

B A D C.

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

Single motor, double current mode of operation

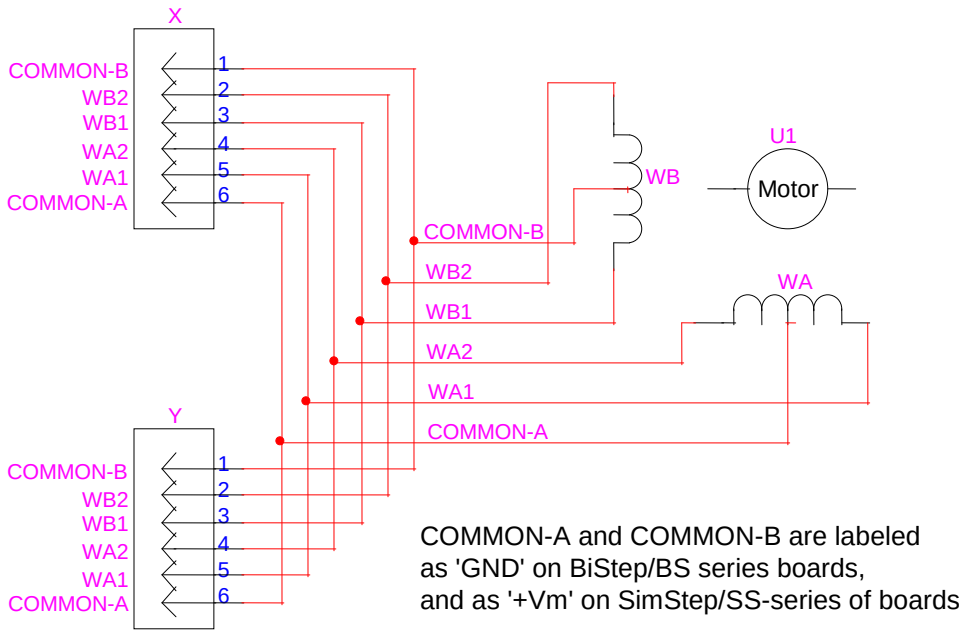
When operating a single motor, it is possible to configure the board to operate that motor at up to 2 times the normal rated current for the board. For example, a single 2 amp motor can be operated by our BS1010 board (which normally has a top current of 1 amp/winding/motor) if the board is configured as described in this section.

You need to 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) through use of the 't' command (Bits 13-15: Motion control configuration: Instant Stop and Double Current Modes). You also need to save that configuration as the RESET/REBOOT default configuration through use of the 'e' command (Saving the current firmware settings).

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); 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.**

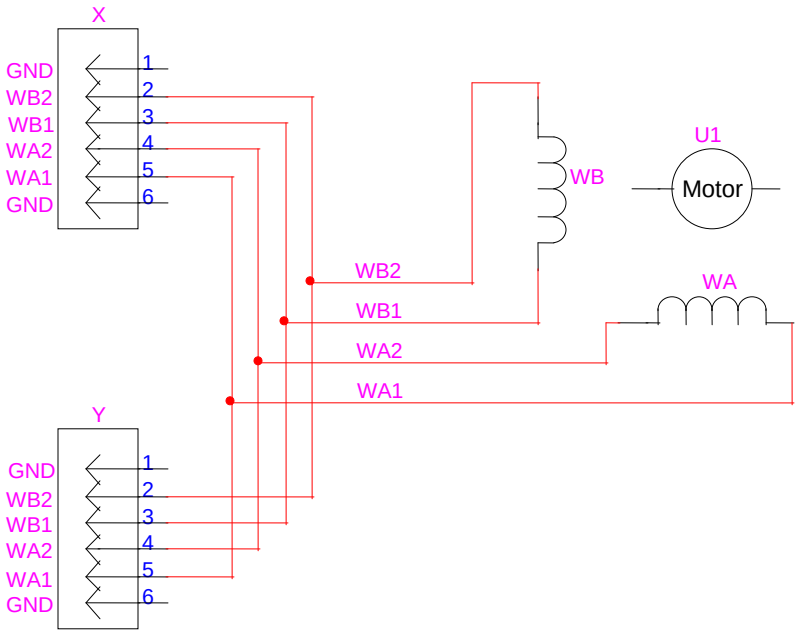
The following two schematics show the wiring for double current mode on both unipolar and bipolar motors.

Wiring a Unipolar motor for double current mode



Unipolar Motor Double Current Mode Connection
To the BS1010 and SS1010 series of boards.

Wiring a Bipolar motor for double current mode



Bipolar Motor Double Current Mode Connection
To the BS1010 series of boards.

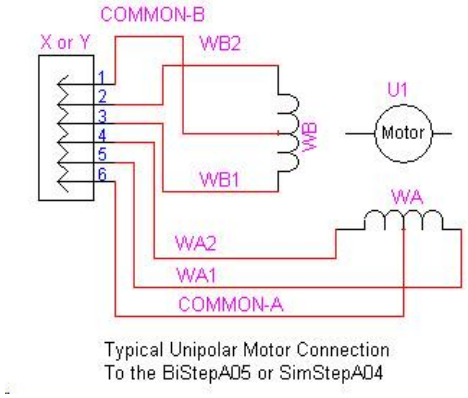
Motor Wiring Examples

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 BS1010 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	BS1010
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	BS1010
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	BS1010
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	BS1010
Black	1
Orange	2
Green	3
Yellow	4
Blue	5
White	6

Jameco 169201 24 Volt, 0.3 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	BS1010
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	BS1010
Red	1
Orange	2
Pink	3
Yellow	4
Blue	5
<no connection>	6

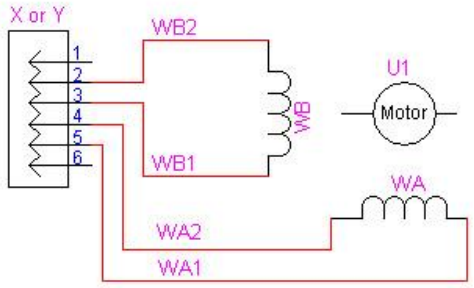
Jameco 174553 12 Volt, 0.6 Amp/winding, 7.5 deg/step

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

Color	BS1010
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	BS1010
<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	BS1010
<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	BS1010
<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 BS1010 to overheat (and probably fail) after just a short period of use, if the BS1010 is configured for the default operation of running two motors at a time. We tested it with the wiring of:

Color	BS1010
<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 jumper the board as described in the 'Configuring Double Current' mode section of this manual, 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!***