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 the SimStep and BiStep based RelayStepper Controllers

By
Peter Norberg Consulting, Inc.
Matches RelayStepper Firmware Revision 3.11

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 RelayStepper version 3.11. The manual version shown on the front page has the same value as the associated RelayStepper version.

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

Version	Date	Description
1.4	September 8, 2003	Initial release
1.5	September 12, 2003	Added sense of 1K resistor between SO and GND as method of setting half-power mode at power on.
1.6	October 29, 2003	Allowed setting the microstep size to 1 full step (prior max was ½ step)
	February 21, 2004	Corrected documentation for 'stop oK' command
2.2	March 20, 2004	Added Potentiometer rate control behavior, for when unit is operated without an external controller, and changed default power-on state to 'Use +/- Y inputs to request motor slew'. **This update is a critical change as compared to version 1.6: harware strap options have changed!**
3.0	May 1, 2004	Adjusted Potentiometer behavior to use a 1.0 uF capacitor instead of a 0.1 uF capacitor to sense the potentiometer setting.
3.1	November 8, 2004	Added automatic sense for presence of the potentiometer, as well notes on special order options
3.4	February 19, 2005	Added order options for limit switch action being 'hard' or 'soft'
3.7	April 18, 2006	Corrected microstep size error
	March 20, 2008	Added notes about surface mount board configuration
3.11	October 31, 2008	Improved serial resynchronization on bad serial data detected

*Please note that version 2.2 is a **major release upgrade** relative to the version 1.6 firmware. The hardware strap options have changed (specifically, 2400 baud communication is no longer controlled by a resistor on the RDY line), and many of the TTL input lines are now "connected" to specific motor and relay behaviors. Please read the "Hardware Configuration" section to get a summary of the new capabilities.*

The RelayStepper firmware allows the current boards in the BiStep and SimStep product lines to simultaneously operate one stepper motor, up to 4 relays, and up to 9 TTL input lines. The main difference between artworks has to do with the current that can be supplied to the relays, and with the types of stepper motors that can be controlled.

Product Safety Warnings

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

- The 7805 5 volt regulator (located near the bottom of the board on most products)
- The L293D power drivers (2 located near the right-hand side of the BiStepA04 and the BiStepA05, normal power version)
- The SN754410 power drivers (these replace the L293D drivers on the BiStepA05 1 Amp option board, and are those used on the BiStepA06 board)
- The circuit board underneath or near the L293D/SN754410 power drivers on the BiStep series of boards (the board itself is used as a heat sink, and hence can become physically hot to touch)
- The L298 drivers and heat sinks on the BiStep2A unit

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

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

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

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

LIFE SUPPORT POLICY

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

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

Introduction and Product Summary

The RelayStepper stepper motor control firmware from Peter Norberg Consulting, Inc., is designed to operate on the SimStep and BiStep series of boards. It converts the board into a tool that can operate up to 4 relays, while operating one stepper motor. If relays (or similar output devices) are to be driven, the drive current available is 0.1 Amps per relay for the SimStepA05 and SS0705 boards. For all of the BiStep series, the current available is ½ of the rated current for the given board. For example, the 1 Amp BiStepA06 unit can support relays (or coils) which require up to ½ amp of drive current.

The TTL input lines can be programmed to be used to manually control slewing of the motor (spinning clockwise and counterclockwise), to provide for limit switch detection, and to directly operate two of the four relays.

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

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

Short Feature Summary

- One stepper motor and up to 4 relays may be controlled at the same time.
- From 2 through 9 TTL input lines are available for monitoring, depending on user-settable parameters.
- For the BiStep product, each motor may be either Unipolar or Bipolar. The SimStep is Unipolar only.
- Each motor may draw up to 0.5 amps/winding on the SimStep series (SS0405, SS0705, SS0805 and SS0905). For the BiStep series (BSxxxx), the standard is that defined by the particular board (0.6, 1.0 and 2.0 amp versions are currently available).
- Limit switches may be used to automatically request motion stop of the motor in either direction.
- Manual slewing of the motor may optionally be requested by the TTL input lines
- The step rate for the motor may be controlled by a user-provided potentiometer
- Two of the relays may optionally be controlled by two of the TTL input lines
- Rates of 1 to 62,500 microsteps per second are supported.
- Step rates are changed by linearly ramping the rates. The rate of change is independently programmed for each motor, and can be from 1 to 62,500 microsteps per second per second.
- All motor coordinates and rates are always expressed in programmable microunits of up to $1/64^{th}$ step. Changing stepping modes between half, full and micro-steps does not change any other value other than which winding pairs may be driven at the same time, and how the PWM internal software is operated.
- Motor coordinates are maintained as 32 bit signed values, and thus have a range of -2,147,483,647 through +2,147,483,647.
- Both GoTo and Slew actions are fully supported.
- Four modes of stepping the motor are supported:
- Half steps (alternates 1 winding and two windings enabled at a time),
- Full power full steps (2 windings enabled at a time)
- Half power full steps (1 winding enabled at a time)
- Microstep (programmable to as small as $1/64^{th}$ steps, using a near-constant-torque PWM algorithm)
- A TTL "busy" signal is available, which can be used to see if the motor is still moving. This information is also available from the serial connection.
- Complete control of the motor, including total monitoring of current conditions, is available through the 2400 or 9600 baud serial connection.
- Runs off of a single user-provided 7.5 to 15 volt DC power supply, or two supplies (7.5-15V for the logic circuits and 7- 26 or 4.5-35V for the motor and relays).
- Any number of motors may be run off of one serial line, when used in conjunction with one or more SerRoute controllers.

Firmware Configuration At Time of Ordering Product

As of version 3.1, the RelayStepper firmware has a set of initial settings that are selected at power-on or reset *that may be reconfigured at the time the product is ordered*. With the exception of the mode of stepping used when the "Auto-full-step" rate is reached, all of these features may be reset through use of the appropriate serial command. Note that firmware version 3.0 uses the "normal" values shown on this page for these features.

Default Microstep Size

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

Default Stop Rate

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

Default Ramp Rate

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

Default Auto-Full-Step Rate

Normally, the firmware defaults to a rate of 3072 microsteps/second as being the rate at which it selects the "Auto-Full-Step" mode (equivalent to the '**3072A**' command). This can be ordered as any rate which is valid for the system.

Default Auto-Full-Step Mode

Our testing of the product shows that once you exceed a given rate (as defined by the 'Auto-Full-Step Rate' command/setting), you can obtain more torque from the motors by switching to simple full-step operation. By default, the "double winding" mode (equivalent to the "**2o**" command) is selected when this "Auto-Full-Step" rate is reached, as that has worked best with the motors that we have tested. However, the mode used may be defined by you at the time of ordering the product to be any of the modes available from the "o" command. Please see the "A" command for details about the "Auto-Full-Step" mode command.

Default Full-Power Level (No 1K resistor on SO, or S1K jumper missing)

Normally, we ship the product such that the default code will select full winding current operation (see the "**0H**" command) when the board is reset or powered on and there is no 1K resistor installed between the SO signal and GND (or, if available on your board, the S1K jumper is removed). At the time of ordering the product, you may change this to operate in ½ power mode ("1H") in this case.

Default Low-Power Level (1K resistor installed on SO, or S1K jumper present)

As with the Full-Power-Level, we also provide an automatic selection of ½ power level (approximately) at the time of board reset (equivalent to the "**1H**" command). This mode may be configured by inserting a 1K resistor (1/4 or 1/8 watt) between the SO TTL output signal and GND (or, if your board supports it, you may install the S1K jumper). You may optionally order this to be the full power level ("0H") if this is better for your application.

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

Default Motor Idle Winding Current

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

Default Limit-Switch Stop Mode

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

Configuring Serial Baud Rate

By default, all serial communications with the PotStepper firmware version 3.0 and higher operate at 9600 baud, 8 data bits, 1 stop bit, no parity. If you need to communicate at 2400 baud, you must order the board configured for the different baud rate.

Hardware Configuration

These notes are valid only for version 2.0 and later of the RelayStepper firmware. Versions 1.0 through 1.6 had different hardware configuration options; please refer to the version of the RelayStepper manual which matches your firmware for more details.

The RelayStepper firmware has two features that can be configured as startup options. This means that any combination of these features may be automatically controlled whenever the firmware receives a power-on, hardware reset, or software reset action. Both features are selected by adding an external 1K resistor to ground a specific TTL output pin (or by equivalent board jumpers, R1K and S1K).

Configuring the default TTL line control mode

The version 2.0 and later firmware supports use of the "Y-" and "Y+" input lines to manually cause the motor to spin left and right, as well as use of the "X-" and "X+" lines to manually control two of the relay lines (XWB1 and XWB2, respectively). By default, this control is **enabled** after a reset or power-on condition; however, if your application requires that these signals **NOT** be "connected" to the motor and relay control system, you may have the firmware startup code disable the feature through installation of a 1K resistor (1/4 or 1/8 watt) between the RDY TTL output signal and GND. If your board has a jumper labeled as 'R1K', then you will get the same effect by installing this jumper.

The exact serial command equivalent for this feature is governed by the '**xF**' command, as described later in this manual.

- Resistor/R1K jumper not present: Same as issuing a "**0F**" command
- Resistor/R1K jumper present: Same as issuing a "**60F**" command

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

This feature is only available in versions 2.0 and later of the RelayStepper firmware.

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

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

This mode may be configured by inserting a 1K resistor (1/4 or 1/8 watt) between the SO TTL output signal and GND. Alternatively, if your board has a jumper position labeled "S1K", you may enable this feature by installing that jumper. The hardware selection may be changed at any time through issuing the "1h" or "0h" commands, as described elsewhere in this manual. However, by operating through use of this hardware strap, you are much less likely to ever "blow out" a board by failing to issue the "1h" command after a power-on or reset condition!

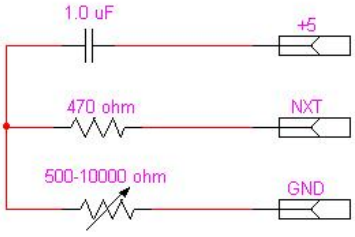
Please refer to the following manual section entitled "Board Input Signals" for information on where to find the SO signal.

This hardware strap is available on firmware versions 1.5 and later.

Potentiometer Control

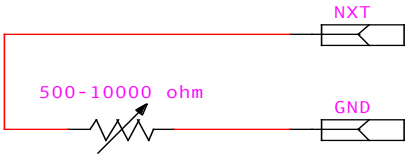
By default, the firmware is set up to expect the target rate to use for the motors to be controlled by a potentiometer connected to the NXT input line. For this to work, a circuit containing a 1.0 uF capacitor, a 470 ohm resistor, and a 500 to 10K ohm potentiometer must be connector to the board. The 1.0 uF capacitor has one side connected to +5 volts. The other side is connected both to the 470 ohm resistor (whose other side is connected to the NXT input on the board) and to the wiper of a 500 ohm to 10K ohm potentiometer. The other side of the potentiometer is connected to ground. *Note: Versions of RelayStepper prior to 3.0 require use of a 0.1 uF capacitor in this circuit.*

The actual schematic for this control is:



Variable Resistor Circuit for use with PotStepper and UCPotStepper firmware. Selection of the resistance of the potentiometer controls the value range available for the resulting rate generated by the firmware.

On our surface mount boards (such as the BS0610G or later, any of the BS0710 series, and the SS07xxUSB series), the board incorporates the above 470 ohm resistor and capacitor as part of its circuit: You only need to add your external 500-10000 ohm variable resistor in between the NXT input and GND. The jumper labeled "PS" must be installed as well (this jumper is located near the 'LIM' and 'IO' connectors, and should have been installed at the factory at the same time that this firmware option was programmed).



Variable Resistor Circuit for use with PotStepper and firmware when installed on any board which has built-in support for the potentiometer (such as the BS0610M). Selection of the resistance of the potentiometer controls the value range available for the resulting rate generated by the firmware.

The firmware uses the time that it takes to discharge the capacitor through the potentiometer to ground as the source of information to use on determining the rate to apply to the motor(s). The rate range is from 1 to XXXX microsteps/second, where XXXX is controlled by the size of the potentiometer.

Resistance (Ohms)	Approximate Maximum Rate (microsteps/second)
500	950
1000	2600
2000	6700
5000	16300
10000	30700

The above numbers are approximate, and may be off by as much as 20%.

By issuing the rate change serial command ("xR", where "x" is the desired rate), each motor may be independently connected or disconnected from the potentiometer. If 'x' is set to 0, then the currently selected motor gets its rate from the current potentiometer position. Otherwise, the requested rate (such as 450 from the command "450r" becomes the new target rate for the given motor.

Power-On (and reset) Defaults

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

- **3072A** – Set the Automatic Full Step rate to be >=3072 microsteps/second
- **0**= – Reset the motor to be at location 0
- **0F** – Enable limit switch detection and motion processing via TTL inputs. If there is a 1K resistor between the RDY output signal and GND (or the R1K jumper is installed), then a "**60F**" command is issued instead (limit switches enabled, motor motion and relay control disabled).
- **0H** – Set motor to full power mode. If there is a 1K resistor between the SO output signal and GND (or the S1K jumper is installed), then the motor is set to ½ power mode (same as the "**1H**" command).
- **80K** – Set the "Stop OK" rate to 80 microsteps/second
- **3O** – Set the motor windings Order to "microstep"
- **8000P** – Set the rate of changing the motor speed to 8000 microsteps/second/second
- **0R** – Set the target run rate to be controlled by the potentiometer
- **1V** – Set <CR><LF> sent at start of new command, no transmission delay time, 9600 baud communications. See the 'V' command to observe how to reset the rate to 2400 baud.
- **0W** – Full power to motor windings, as defined by the 'H' command.

Board Input Signals

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

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

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

Pin	Board Label	Signal as used in this manual	Signal Description
1	RTC		<not to be used>
2	+5		+5 volts
3	RST		Board Reset
4	GND	GND	Ground
5	GND	GND	Ground
6	A0	LY-	Y- limit input
7	A1	LY+	Y+ limit input
8	A2	LX-	TTL input
9	A3	LX+	TTL input
10	B0	Y-	Y- slew request
11	B1	Y+	Y+ slew request
12	B2	X-	Controls XWB1 relay
13	B3	X+	Controls XWB2 relay
14	B4	NXT	Potentiometer rate control
15	B5/READY	RDY	Motor Ready Output
16	B6/SERIN	SI	TTL Serial Input
17	B7/SEROUT	SO	TTL Serial Output
18	+5		+5 volts
19	GND	GND	Ground

Release 2 – BiStepA06, BS0610, BS0710, BiStep2A, SS0705, SS0805, SS0905

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

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

TTL Input Voltage Levels

All TTL input signals are treated as CMOS levels. This means that a logic "0" is generated at any time that the input voltage is <= ½ of the board power, and a logic "1" is generated when the input voltage is above ½ of the board power. Therefore, since our power is 5 volts, a logic "0" is presented when the input is <= 2.5 volts, and a "1" is presented when the signal is above 2.5 volts. In reality, we suggest using <=2 volts for a "0", and >=3 volts for a "1", to avoid any "noise" issues.

Note also that when performing normal TTL input on the non-limit switch signals (those available on the SLEW and IO connector), the firmware performs a digital filter on the input signal, to remove possible noise spikes.

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

On the later boards (SS0805, SS0905, BS0610 revision N and above, and the BS0710), there is a 1K pullup resistor installed on all of the TTL input signals (the limits and the slews). This improves the noise immunity of the inputs, and allows use of optical sensors, since more current is available as part of the input system.

Input Limit Sensors and TTL inputs, lines A0/LY- to A3/LX+

Lines A0/LY- through A3/LX+ are used by the software to either limit the motor motion (on the Y connector), or as generic input TTL signals (either as standard direct output from a TTL device, or as switch closures to ground). At power on, the system defaults to using LY- and LY+ as motion limits; however, through use of the "F" command, the firmware may be told to simply treat all of the inputs as purely informational.

The connections are:

Signal	***Use***
A0/LY-	-Y limit
A1/LY+	+Y limit
A2/LX-	TTL input or pulse source for pulse counter
A3/LX+	TTL input

The connections may be implemented as momentary switch closures to ground or as TTL signals; on most connectors, a ground pin is available just above the A0/LY- pin. They are fully TTL compatible; therefore driving them from some detection circuit (such as an LED sensor) will work. The lines are "pulled up" to +5V with a very weak (10-20K) resistor, internal to the SX-28 microcontroller. On the later boards (SS0805, SS0905, BS0610 revision N and above, and the BS0710), there is a 1K resistor pullup installed on all of these lines, thus increasing the noise immunity of the circuit.

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

As the most extreme example possible:

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

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

Also note that use of the "!" emergency reset command, 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!***

Note that as of version 3.4, it is possible to order the firmware configured for "instant stop" on the limit switches. As with the '!' command, if the firmware is configured with this mode of operation, ***please be aware that, in some designs, damage to gear systems can result when such a sudden stop occurs. Use this feature with care!***

Motor Slew Control and more TTL inputs: B0/Y- to B5/RDY

Lines B0/Y- through B4/NX are used to control stepping of the motor, and the rate of steps. The inputs are normally fully debounced and are designed to operate via a microswitch closure to ground or TTL levels of voltage input.

The connections are:

Signal	***Action Requested***
B0/Y-	TTL input or Slew -Y
B1/Y+	TTL Input or Slew +Y
B2/X-	Controls XWB1 relay
B3/X+	Controls XWB2 relay
B4/NXT	Potentiometer Rate Control
B5/RDY	Motor Ready (output signal)

All of the input signals on this connector are filtered through a software debounce routine before they are acted on, in order to reduce false triggers.

When operated with the slew inputs enabled (see the 'F') command, 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 (that is to say, B0/Y- and B1/Y+ both being low at the same time) are treated as "stop", to avoid confusion. As with all other operations of the system, the motor is accelerated to the current rate using the ramp rate defined within the code (which defaults to 4000 microsteps/second/second).

The "X-" and "X+" inputs may be used to control the XWB1 and XWB2 relays, respectively. As long as they have not been disabled by the 'F'eatures command (or the 1K hardware strap on RDY), changing either (or both) of these inputs will cause the associated relay to close (when the input goes low) or open (when the input goes high). Note that serial control of the relays (for example using the 'C' or 'D' commands) may operate at the same time as the TTL commands. The firmware simply executes the "last" command that it has seen for a given relay. For example, closing the "X-" input (shorting it to ground) will cause the relay connected to XWB1 to be actuated. If the board then receives a "4U" command, that relay will be opened (even though the TTL input is still closed).

The "Potentiometer Rate Control" allows for a user-provided potentiometer to be used to set the step rate for the motor. Please see the earlier section of the manual entitled "Potentiometer Rate Control" for details of this feature.

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

The B5/RDY output signal is designed to indicate whether the motor is still in motion. When HIGH, then all motion is stopped. When LOW, the motor is still moving.

Raw serial I/O

These signals give access to the serial I/O for the SX-28, and are used when the MAX232 chip (or, if present, the 'JS' jumper) is removed.

Name	**Description**
B6/SERIN/SI	INPUT: Raw SX-28 Serial Input (TTL level)
B7/SEROUT/SO	OUTPUT: Raw SX-28 Serial Output (TTL Level)

SI and SO are the "real" serial input and serial output (respectively), as seen by the SX-28 chip. If the application desires direct serial communications without RS232 levels (for example, if the Parallax Inc.[tm] Basic Stamp [tm] based products are being used to control the board), simply remove the MAX232 chip (or, if present, remove the 'JS' jumper) and use these pins. **Similarly, the MAX232 chip (or the equivalent 'JS' jumper) must be removed if this board is a "child" board in a SerRoute-based tree of boards.**

Serial Operation

The RS-232 based serial control of the system allows for full access to all internal features of the system. It operates at 2400 or 9600 baud, no parity, and 1 stop bit. Note that you should wait about ¼ second after power on or reset to send new commands to the controller; the system does some initialization processing which can cause it to miss serial characters during this "wake up" period. In order to operate at 2400 baud, you must first send a 'V' command at 9600 baud to tell the board to switch to 2400 baud operation (see the 'V' command documentation, later in this manual).

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

Serial Commands

The serial commands for the system are described in the following sections. The code is case-insensitive (i.e., "s" means the same thing as "S"). ***Please be aware that any time any new input character is received, any pending output (such as the standard "*" response to a prior command, or the more complex output from a report) is cancelled.*** This avoids loss of commands as they are being sent to the control board.

Serial Command Quick Summary

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

0-9, +, - – Generate a new VALUE as the parameter for all FOLLOWING commands
A – Select the Auto-Full Power Step Rate
C – Close (actuate) selected relays
D – Set all 4 relays open or closed at once
F – Set board features
G – Go to position x on the motor
H – Operate motor at ½ power
I – Wait for motor 'Idle'
K –Set the "Stop oK" rate
L – Latch Report: Report current latches, reset latches to 0
M – Mark location, or go to marked location
O – step mOde – How to update the motor windings
P – sloPe (number of steps/second that rate may change)
Q – Pulse count on line LX-
R – Set run Rate target speed for the motor
S – start Slew
T – limiT switch control
U – Open (release) selected relays
V – Verbose mode command synchronization
W – Set windings power levels on/off mode for selected motor
Z – Stop motor
! – RESET – all values cleared, motor set to "free", redefine microstep. Duplicates Power-On Conditions!
= – Define current position for the motor to be 'x', stop the motor
? – Report status
other – Ignore, except as "complete value here"

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

Possible combinations:

"-" alone – Set '-' seen, set no value yet

"+" alone – Clear '-' seen, set no value yet

-n: Value is treated as -n

n: Value is treated as +n

+n: Value is treated as +n

Examples:

-1? – Request report on current relay settings

3F – Set input control masks to disable limit switches

xA – Select the Auto-Full Power Step Rate, to increase top motor speed

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.*

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

For example,

3072A

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

Set this to 62500 to disable this feature.

xC – Close (actuate) selected relays

This sets the requested relay bits to '1', thus closing the indicated relay(s). The relays are assigned one per bit, with the relay number matching the bit value (counting from 0). Simply form the correct sum from the following table, to tell the code which set of relays to close. The relays are connected to the X motor connector as shown in the following table

Relay	Add Value	X connection
0	+1	WA1
1	+2	WA2
2	+4	WB1
3	+8	WB2

For example,

5C

will close relays 0 and 2 if they are not already closed. No other relays will be affected.

xD – Set all 4 relays closed (1) or open (0) as requested.

This sets all relays to the values shown, thus opening and closing all relay. . The relays are connected to the X motor connector as shown in the following table

Relay	Add Value	X connection
0	+1	WA1
1	+2	WA2
2	+4	WB1
3	+8	WB2

For example

7d

will close relays 0, 1 and 2; and will open relay 3.

xF – Feature control: limit switches and slew interpretation

The Feature command is used to control interpretation of the board limit switch and slew request inputs. **By default (after power on and after any reset action), the board is configured to respond to each of the Y limit switches.** 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.

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

Bit	Numeric Sum Value	Block Switch Detection On or disable connection to…
0	+1	LY- (limit)
1	+2	LY+ (limit)
2	+4	Y- (slew)
3	+8	Y+ (slew)
4	+16	X- 'connected' to XWB1 relay
5	+32	X+ 'connected' to XWB2 relay

The "X-" and "X+" inputs may be used to control the XWB1 and XWB2 relays, respectively. As long as they have not been disabled by the 'F'eatures command (or the 1K hardware strap on RDY), changing either (or both) of these inputs will cause the associated relay to close (when the input goes low) or open (when the input goes high). Note that serial control of the relays (for example using the 'C' or 'D' commands) may operate at the same time as the TTL commands. The firmware simply executes the "last" command that it has seen for a given relay. For example, closing the "X-" input (shorting it to ground) will cause the relay connected to XWB1 to be actuated. If the board then receives a "4U" command, that relay will be opened (even though the TTL input is still closed).

For example,

4f

would block detection of the "Y-"slew request, and allow all of the other switches to work as normal. Similarly,

63f

would block detection of all limits, slews, and would disable TTL control of the two relays, and

0f

would enable all switch actions.

Version 2.0 of the firmware acts as if a "0F" command has been given any time that a power-on or reset condition occurs. This means that the LY- and LY+, Y- and Y+ inputs are all actively processed.

Note that versions of the firmware before 2.0 have a different power-on default state, which disables the Y- and Y+ slew inputs as well as the 'Nxt' rate control (bit number 4). Please refer to the 1.x RelayStepper manual for more information about operation of the 'F' command in the older firmwares.

xG – Go to position x on the Y motor

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

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

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

For example,

1000g-25687g

Would:

1. Start a GOTO on motor Y to location 1000
2. Start a GOTO on motor Y to location –25687, aborting the goto location 1000

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

-1?

Could report

Y,-1,350

*

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

xH – Operate motor at ½ power

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

The two settings for this are:

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

1H – Run in ½ power mode

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

The firmware can automatically select the initial state of the ½ power mode by sensing the presence of a 1K resistor between the SO (Serial Output) signal and ground (or the presence of the S1K jumper). If there is no resistor (S1K) jumper) there, then full power mode will be selected at power-on or reset. However, if there is a 1K resistor between SO and GND (or the S1K jumper is present), then the firmware will select ½ power mode as the initial state, thus avoiding potential damage to the controller.

I – Wait for motor 'Idle'

This allows your code to 'wait' for the Y motor to be idle. The code simply waits for the Y motor to complete its motion 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 Y location, and then wait for the motor to actually get there, you could simply issue the command sequence:

Send	Receive
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).

xK –Set the "Stop oK" rate

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

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

For example,

100k

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

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

62500K

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

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	Reserved—not currently used
3	+8	Reserved—not currently used
4	+16	System power-on or reset ("!") has occurred

For example, after initial power on,

```
L
```

Would report

```
L,16
*
```

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

```
L,4
*
```

xM – Mark location, or go to marked location.

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

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

x=1 : Go to last "marked" location

xO – 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 1/64th step, pwm-estimated 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).

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

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

The slope can be specified to be from 1 through 62,500 microsteps per second per second. If a value of 0 is specified, the code forces it to have a value of 8000. If a value above 62,500 (or less than 0) is specified, the code will accept it, but will ramp unreliably (i.e., do not do it!).

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

For example, if the motor is currently at location 0, then the sequence:

250p500r2000g

would cause the following actual ramp behaviors to occur:

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

xQ – Perform pulse counting on line A2/LX-

The 'Q' command causes the code to both set a sample period (to 'x' milliseconds), and to actually count the number of edges (low-to-high and high-to-low) which occur on the TTL input line A2/LX-. The code operates as follows:

1. Interpret the 'x' parameter passed. If it is 0, use the prior sample period (which defaults to 10 milliseconds on power on); otherwise, use the value passed as the desired number of milliseconds to count pulse edges.
2. Clear the count of pulses seen.
3. Send a 'Q' back to the host, to tell it that counting has started.
4. Start counting pulses, and continue counting for the specified number of milliseconds.
5. If a new character is received before the sample period has completed, abort the request.
6. Otherwise, report the total number of EDGES seen during the time interval.

The number returned will be about 2x the "frequency" of the pulses, since both the leading and trailing edges are counted. For example, if you have a 1 kHz input signal, and you request a "10q" to sample for 10 milliseconds, the value reported will be 20 counts.

For example,

10q

could report

Q,20

On a 1 kHz input signal. Note also that issuing a

0Q

after having issued a previous "1000q" would use the 1000 milliseconds (1 second) as its sample period.

The the minimum detectable pulse width is 8 microseconds. This means that the code cannot detect more than 125,000 edges per second; and that input frequencies which exceed 62,500 Hz will not be correctly interpreted.

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

This defines the run-rate to be used for the motor. It may be specified to be between 0 and 62,500 microsteps per second. If a value of 0 is specified, the code connects the motor to the rate-potentiometer (see the hardware discussion at the start of this manual). If a value outside of the limits is specified, then it is accepted, but the code will not operate reliably. Do not specify values outside of the 0-62,500 legal domain.

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

For example,

250R

Sets the motor target stepping rate to 250 microsteps per second.

The power-on/reset default Rate is 0, which connects the potentiometer to the rate.

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.

As another example, issuing the command

0R

will cause the motor to get its rate information from the potentiometer circuit.

xS – start Slew.

The "S"lew command is used to cause the 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

xU – Open (release) selected relays

This sets the requested relay bits to '0', thus opening the indicated relay(s). . The relays are connected to the X motor connector as shown in the following table

Relay	Add Value	X connection
0	+1	WA1
1	+2	WA2
2	+4	WB1
3	+8	WB2

For example,

10u

will open relays 1 and 3 if they are not already open. No other relays will be affected.

xV – Verbose mode command synchronization

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

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

The verbose command is bit-encoded as follows:

Bit	SumValue	Use When Set
0	+1	Send <CR><LF> at start of processing a new command
1	+2	Delay about 1 character time before transmission of first character of any command response. 2 more stop bits are also added to each transmitted character, to allow more processing time in the receiving microprocessor.
2	+4	Controls serial BAUD rate. 0 means communicate at 9600 baud (the default), while 1 means communicate at 2400 baud

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

For example,

0v

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

3v

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

The complete table of options is:

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

Operating at 2400 Baud

In order to operate the controller at 2400 baud, you must issue a 'V' command at 9600 baud to change the rate to 2400 baud. All future 'V' commands must include the "operate at 2400 baud" flag (bit 2), or the board will revert to its standard 9600 baud action.

For example: After power on or reset, the board is always initialized as if a '1V' command has been issued ("Operate at 9600 baud, send <CR><LF> after each command is received"). In order to set the mode to "Operate at 2400 baud, no <CR><LF>, add extra delay time to the response" you would:

1. Set YOUR communications to transmit at 9600 baud
2. Send "6V", to select the new baud rate and delay time
3. Set YOUR communications to operate at 2400 baud.
4. From now on, all characters should be sent and received at the new baud rate

xW – Set windings power levels on/off mode for the Y motor

The "W"indings command controls whether the Y motor 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 motor is 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.

Z – Stop motor

'Z' causes the motor 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,

```
z
```

Would slow down, then stop motor Y.

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

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

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

The suggested values would be the powers of 2, vis. 1, 2, 4, 8, 16, 32 and 64 (giving you true microstep step sizes of 1/64, 1/32, 1/16, 1/8, ¼, ½ and 1, respectively). All other values (such as RATE or GOTO LOCATION) are then expressed in units of the microstep size; therefore, location "3" would mean "3/64" in the finest resolution (microstep set to 1), and "3" in the largest resolution (microstep set to 64). Note: versions of firmware prior to 1.6 allowed up to ½ step as being the largest microstep size (i.e., a value of 32 for this function).

For example,

4!

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

The reset command also selects the following settings:

- **3072A** – Set the Automatic Full Step rate to be >=3072 microsteps/second
- **0**= – Reset the motor to be at location 0
- **0F**– Enable limit switch detection and motion processing via TTL inputs. If there is a 1K resistor between the RDY output signal and GND (or the R1K jumper is installed), then a "**60F**" command is issued instead (limit switches enabled, motor motion and relay control disabled).
- **0H** – Set motor to full power mode. If there is a 1K resistor between the SO output signal and GND (or the S1K jumper is installed), then the motor is set to ½ power mode (same as the "**1H**" command).
- **80K** – Set the "Stop OK" rate to 80 microsteps/second
- **3O** – Set the motor windings Order to "microstep"
- **8000P** – Set the rate of changing the motor speed to 8000 microsteps/second/second
- **0R** – Set the target run rate to be derived from the potentiometer setting
- **1V** – Set <CR><LF> sent at start of new command, no transmission delay time, 9600 baud communications. See the 'V' command to observe how to reset the rate to 2400 baud.
- **0W** – Full power to motor windings, as defined by the 'H' command.

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

This copies the current VALUE as the current position for the motor, and then stops said motor. For example,

2000=

Would define the current location to be 2000. 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.

2000=g

Would define the current location of the 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 the motor, or about internal states of the software.

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

1-255: Report memory location 1-255

Useful locations:

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

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

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

- -1; Report current location
- -2; Report current speed
- -3; Report current slope
- -4; report target position
- -5; Report target speed
- -6; Report windings state
- -7; report stop windings state
- -8; Report step action (i.e., motor state)
- -9; Report step style
- -10; Report run rate
- -11; Report stop rate
- -12; Report current software version and copyright
- other: Treat as 0 (report all except version/copyright)

All of the reports follow a common format, of:

1. If Verbose Mode is on, then a <carriage return><line feed> ("crlf") pair is sent.
2. The letter corresponding to the motor being reported on is sent (i.e., 'Y').
3. A comma is sent.
4. The report number is sent (such as –4, for target position).
5. Another comma is sent.
6. The requested value is reported.
7. If Verbose Mode is on, then a <crlf> is sent
8. A "*" character is sent.

The special reports which are understood are as follows:

0: Report all reportable items

The "report all reportable items" mode reports the data as a comma separated list of values, for reports –1 through –11. Just after power on, for example, the request of "0?" would generate the report:

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

Where:

- Y is the motor
- 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,

```
0?
```

Would report all reportable values for the motor. You could receive:

```
Y,0,30,10,1000,30,10,0,0,0,1,100,10
*
```

-1: Report current location

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

For example,

```
-1?
```

Would report the current location on the motor. You could receive:

```
Y,-1,25443
*
```

-2: Report current speed

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

For example,

```
-2?
```

Would report the current speed on the motor. You could receive:

```
Y,-2,2502
*
```

-3: Report current slope

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

For example,

```
-3?
```

Would report the current rate on the motor. You could receive:

```
Y,-3,25443
*
```

-4: Report target position

This reports the target location for the motor.

For example,

```
-4?
```

Would report the current target on the motor. You could receive:

```
Y,-4,-35443
*
```

-5: Report target speed

This reports the current target run rate which is desired for the motor. 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,

```
-5?
```

Would report the target rate on the motor. You could receive:

```
Y,-5,250
*
```

-6: Report windings state

This reports the current energized or de-energized state for the windings for the motor. A reported value of 0 means "the windings are off", a value of 1 means "the windings are energized in some fashion".

For example,

```
-6?
```

Would report the current state on the motor. You could receive:

```
Y,-6,0
*
```

-7: Report stop windings state

This reports whether the windings will be left energized when motion completes for motor. 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,

```
-7?
```

Would report the requested state on the motor. You could receive:

```
Y,-7,0
*
```

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

This reports the current (instantaneous) state for the motor. 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,

```
-8?
```

Would report the current location on the motor. You could receive:

```
Y,-8,4
*
```

This would mean that 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 motor. 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

For example,

```
-9?
```

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

```
Y,-9,2
*
```

This would equate to the Y motor running in full-power, full step mode.

-10: Report run rate

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

For example,

```
-10?
```

Would report the current rate on the motor. You could receive:

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

-11: Report stop rate

This reports the speed at which the motor may be considered to be stopped, for starting and stopping activities for the motor.

For example,

```
-11?
```

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

```
Y,-11,50
*
```

-12: Report current software version and copyright

This reports the software version and copyright.

For example,

```
-12?
```

could report:

```
*
relaystepper.src $version: 1.3$
Copyright 2003 by Peter Norberg Consulting, Inc. All Rights
Reserved
*
```

other – Ignore, except as "complete value here"

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

123 456G

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

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

Basic Stamp™ Sample Code

The StepperBoard series of boards may all be used with the Parallax, Inc.tm Basic Stamp tm series of boards. The connection to the StepperBoard product is usually via three of the pins on the IO connector (RDY, SI, and SO), with the MAX232 IC removed from its socket (or the "JS" jumper removed, if available). The remaining input pins may be wired or not, as needed by the application. Most of the time, they will be left unconnected (to "float").

Communications between the two boards may be performed either at 9600 baud (the default), or 2400 baud (via a configuration option). Normally, operating at the 9600 baud rate is recommended; use the 2400 baud rate only if you cannot make your code work at 9600 baud. You **must** use the 'V'erbose command to configure the controller to pause one character time before sending responses to the Basic Stamp, to avoid data synchronization issues. To operate at 2400 baud, you start at 9600 baud and send the 'V' command, telling it to operate at 2400 baud for all future commands.

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

- RDY connected to P3
- SI connected to P2
- SO connected to P1

"RelayStepperTest" is a 9600 baud demo, which uses the serial line for synchronization. It cycles the relay driver outputs through all possible 16 combinations, as well as showing motor motion under the various stepping modes.

The complete sources to this example is installed by default into the "StepperBoard" directory within your "My Documents" folder when you install the code provided with the product.

Listing for RelayStepperTest.bs2 – 9600 Baud test

```
' ************************************************************************
' $modname: RelayStepperTest.bs2$
' $nokeywords$
' Demonstates some of the serial commands using seek and serial response to
' the SimStep and BiStep set of controllers using the RelayStepper firmware
' from Peter Norberg Consulting, Inc.
'
' This source's action is:
'  1. Establish connection to BiStepCom
'  2. Set to allow the controller to take 1 minute on long actions
'   3. Accepts default settings for the controller
'   4. Performs the following test sequence forever
'       a) For each possible relay combination
'               (0 through 15, giving 16 combinations)
'       b)  Set the relays to the requested pattern
'       c)  Set the stepping mode to 3 & (pattern),
'               thus cycling through all 4 stepping modes
'       d)  Seek to motor location 100 * (pattern)
'       e)  Wait for motor idle
'
' Note that, due to changing the winding update order, the sound
' generated during the steps changes.  This is due to the different
' amounts of torque being applied to the motor causing the actual
' motion to be smoother or noisier; the quietest will usually be mode 3
' (microstepping), the noisiest will usually be mode 2 (2-winding full step).
'
' All three modes of stepping are cycled:
'   Mode    Use
'    0      Single Winding mode (1/2 power full steps)
'    1      Half step mode (alternate single/double windings on)
'    2      Full step mode (double windings on)
'    3      Microstep mode (full microstep processing; DEFAULT MODE)
'
'
' SPECIAL TIMING NOTE: It can take the SimStep/BiStep up to 100 uSeconds to
' respond to a new serial "go" command (goto or slew); therefore, you must
' always wait a small amount of time (at least a few milliseconds uSecs)
' before testing the "busy" line, since you may get a "false idle" response.
'
' Since this is a relative seek on the Y motor, you can test the limit switches
' easily;  just ground one of the limit inputs (LY- or LY+) at a time,
' and observe that the Y motor stops going in the indicated direction
'
'       Ground   Direction
'       Line     Blocked
'       LY-/A0      -Y
'       LY+/A1      +Y
'
' ************************************************************************

'
' Be Sure to select the correct stamp and baud rate
' You might have the BS2, BS2SX, etc.
' {$STAMP BS2SX}

' Select the baud rate control:
'  Stamp    9600  2400
'  ----------  ----    ----
'  BS2/BS2e    84   396
'  BS2sx/BS2p   240  1021

PortStepperBaud     CON 240      ' Baud rate to generate 9600 baud on BS2sx

' ************************************************************************


' SimStep or BiStep controller connected as follows.  Please remember to
'   remove the "JS" jumper (or the MAX232 or equivalent serial buffer chip)
'   from the controller, so that the SI line will not be overloaded.

'   Serial Input P1 to controller SO/B7 Serial output
```

```
'   Serial Output p2 to controller SI/B6 Serial Input
'   busy          p3 to controller RDY/B5 Status Output
'                    (HIGH = idle, LOW = motion in progress)

PortStepperSerFrom  CON  1      ' Serial from stepper port
PortStepperSerTo    CON  2      ' Serial to stepper port
PortStepperBusy     CON  3      ' Busy line

PortStepperBusyTest VAR IN3    ' Same as PortStepperBusy, used for input test

idRelayPattern VAR Byte           ' Gets relay pattern; cycles 0 to 15
'szSerString var byte(2)      ' Only used if you enable debug mode (see comments)


' Code restarts here if RESET button pressed

        INPUT        PortStepperBusy                    ' BUSY from stepper

        ' Wait for stepper power on cycle
        PAUSE 250

        ' Reset the stepper, set 1/16 full-step step size
        SEROUT PortStepperSerTo,PortStepperBaud,["4!"]

        ' Wait for stepper to send its wake-up copyright text
        PAUSE 1000

        ' Turn off verbose responses, add delay
        serout PortStepperSerTo,PortStepperBaud,["2V"]

        ' Start with all relays off, at location 0
        idRelayPattern = 0


loop:
        ' Set relay pattern
        SEROUT PortStepperSerTo,PortStepperBaud,[DEC idRelayPattern,"D"]

        ' Set winding update order
        SEROUT PortStepperSerTo,PortStepperBaud,[DEC (idRelayPattern & 3), "o"]

        ' GoTo the value*100
        SEROUT PortStepperSerTo,PortStepperBaud,[DEC idRelayPattern,"00g"]

        GOSUB WaitReady                                    ' Wait until ready

        idRelayPattern = (idRelayPattern + 1) & 15         ' Cycle step type
        GOSUB WaitReady                                    ' Wait until ready
        GOTO        loop                                   ' Cycle forever

WaitReady:
'        DEBUG "Waiting..."

        ' wait for ready; the leading 0's flush BiStep's output queue
        SEROUT PortStepperSerTo,PortStepperBaud,["00I"]

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

SerTest.exe – Command line control of stepper motors

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

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

```
SerTest Text1 Text2 … Textn
```

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

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

  For example,

  /b9600 sets 9600 baud,

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

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

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

  /i60000 – Set 1 minute before timeout

  /i10000 – set 10 seconds before timeout

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

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

  SerTest /pCOM2 4!1000g

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

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

For example,

Sertest 4!1000gi-2000gi

Would:

1. Operate at 9600 baud on COM1 using a 1 minute time out
2. reset the board to operate with a microstep size of 4/64
3. tell the Y motor to go to location 1000,
4. wait for the Y motor to arrive there,

5. tell the y motor to go to location -2000,
6. and wait up to 60 seconds for the motions to complete

Similarly,

SerTest /pCOM2 /b2400 /i10000 +5000s

Would:

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

StepperBoard.dll – An ActiveX controller for StepperBoard products

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

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

Y Connector, Wiring the Motor

Please refer to the UniversalStepper manual associated with your board for information about wiring your motor to the Y connector. Note that when using SimpleSerial to cause your motor to spin as part of the test, you only need to issue the "+s" and "-s" commands.

X Relay Connector, Wiring the Relays

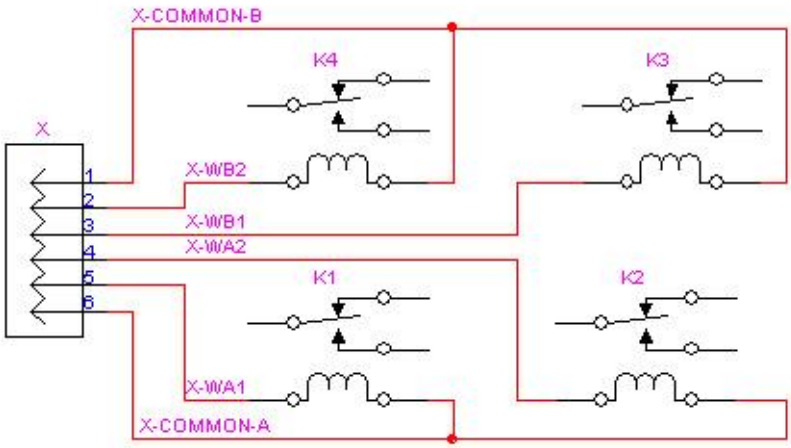
The X connector is used to operate up to 4 relays. One line from the relay is connected either to ground (if using a BiStep-series board as the router) or to one of the +V pins provided as part of the X and Y connectors (if using a SimStep or SS0705). The other line is connected to the appropriate pin of the connector, from the table below.

At power on and after any reset, the relays are configured for "safe" operation. This means that all of the relays are left off at power on.

The complete set of signal assignments for the X connector wired to relays is:

Signal	***Relay Bit***	***Relay Value***
X-WA1	0	+1
X-WA2	1	+2
X-WB1	2	+4
X-WB2	3	+8

Schematically, these may be wired on the BiStepA05 or SimStepA04 as follows:

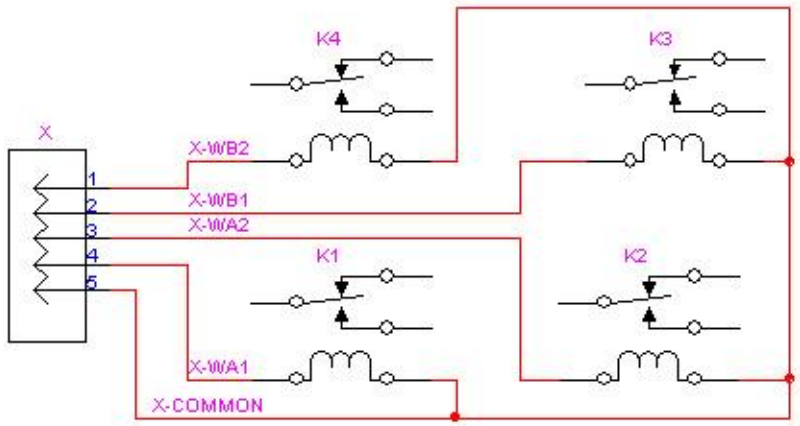


Typical Set of 4 Relay Connections
Using the BiStep or SimStep Series Products
(except for the BiStepA04)

The BiStep A04 board has one fewer ground pin per X, Y connector. The "top" ground pin is deleted (the connectors are 5 pins, not 6), generating the pinout of:

Pin	Name
1	WB2
2	WB1
3	WA2
4	WA1
5	GND

Schematically, these connectors may be wired as follows:



Typical Set of 4 Relay Connections
Using the BiStepA04

Kit Assembly Instructions – Please see the "UniversalStepper" Manual

Please refer to the UniversalStepper manual specific to your product for assembly instructions.