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 SerSeroute Controllers

By
Peter Norberg Consulting, Inc.
Matches SerRoute Firmware Revision 2.5

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 SimStepA04, BiStepA04, and BiStepA05. Newer artworks (such as the SS0705, BiStepA06 and the BiStep2A) are similar; contact the factory for more information if you are using these newer boards. The firmware release described is SerRoute version 2.1. The manual version shown on the front page has the same value as the associated SerRoute version.

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

Version	Date	Description
1.4	7/31/2002	Initial release; cleanup serial details
1.6	1/8/2003	Added 'P' simplified pulse-counting mode
1.8	5/12/2003	Changed TTL input sensitivity from Schmitt Triggered to CMOS (<2 volts=0, >3 volts=1)
1.9	8/17/2004	Added 'L' command for Latched Data read; used to see if a Reset has occurred.
1.10	December 11, 2004	Minor code tweaks; corrected table in manual describing all board connections
1.14	November 3, 2005	Added new 'Pulse Width' measurement option to the 'pulse count' subsystem, extended the allowed set of pulse input ports to be all four 'limit' lines (A0-A3).
1.17	November 16, 2005	Added new 'Encoder' counter subsystem to allow up to 2 phase-encoders to be monitored by the system on lines A0 to A3.
2.0	December 6, 2005	Major upgrade to allow for up to 8 encoders; changes pulse timing
2.1	July 17, 2006	Upgrade to support binary data transmission
	January 10, 2007	Corrected timing notes for the 'P' command
2.5	March 13, 2009	Added 'I' command, for retrieval of latched edge events on the LIM and SLEW inputs

The SerRoute firmware allows the SimStepA04 and SS0705 artworks to control from 3 to 9 child boards, and it allows the BiStep based artworks to control 3 to 5 child boards. The main difference between artworks has to do with whether the relay control can be switched over into additional serial channels, and with the current which can be supplied to the relays.

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 relays 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 relays can become extremely hot during standard usage – some relays 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 **SimStep** and **BiStep** SerRoute serial routing devices from Peter Norberg Consulting, Inc., have similar performance specifications when controlled via the SerRoute firmware. The differences are related to base board power requirements, whether the unit can support at most 5 or 9 child boards, and on the maximum recommended drive current which is available for the relay outputs.

- The SimStepA04 and SS0705 units can be ordered in such a way as to permit up to 9 child boards to be connected to them (basically, the motor driver chip is removed, and either a shunt or a resistor pack is put in its place). All of the BiStep series of boards can support at most 5 child boards.
- 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 SerRoute product, when implemented on the SimStepA04 or SS0705 boards, provides for routing of serial data between one host and 1 to 9 (special order option) "child" boards, each of which may be a SimStep, BiStepA04, BiStepA05, or another SerRoute board. The variance in the number of serial lines comes from the ability of the board to redefine the use of all of its I/O ports from TTL, ENCODER and RELAY control to being TTL-serial interfaces. When implemented on the BiStep series of boards, 1 to 5 "child" boards may be controlled, and larger relays may be used.

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 to Five (or Nine, if a SimStep board is used as the controller and a special order option is chosen) child boards may be directly controlled by one SerRoute board.
- Each child board may be any one of our SimStep, BiStep, or SerRoute products.
- Due to the nested nature of the routing system, there is no physical upper limit to the number of boards which may be run via one serial line; thus, an unlimited number of motors may be controlled via one serial port.
- Each SerRoute board may optionally control up to eight relays. If the relay mode is enabled, then each relay may draw up to 0.1 amps when the SimStep or SS0705 is used as a router. For the BiStep, each relay may be ½ of the rated current of the board (this can require a user-supplied external cooling fan in some cases).
- Each router may be configured to have 4 TTL inputs (this is toggled between having 4 TTL/Encoder inputs or 2 extra serial I/O lines)
- If TTL inputs are permitted, then TTL input lines 0 through 3 may be optionally used as simplified pulse-counting (actually, an edge-counting) and pulse width (version 1.17 and later) measurement inputs.
- Routing commands may be sent either as an ASCII-readable stream of text (such as "{21}", or as binary encoded data (0xF2 0xE1). Both of the above examples would route future serial I/O to board 2 on the controller closest to the host, and board 1 on that selected controller.
- 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 35V for the relays).
- For versions 1.14 and later, if pulse counting is used, you have the option of either counting the number of pulse edges in a given amount of time (equivalent to counting pulses), or of measuring the exact widths of the high and low portions of the pulse (in 8 microsecond units)
- For versions 1.17 and later, you may simultaneously monitor one or two phase-encoders attached to the A0 to A3 inputs. This allows you to keep track of motor position in situations where encoders are available.
- For versions 2.0 and later, up to 8 phase encoders may be attached.

Firmware Configuration At Time of Ordering Product

As of version 1.17, the SerRoute 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*. All of these features may be reset through use of the appropriate serial command. Note that firmware versions prior to 1.17 use the "normal" values shown on this page for these features.

Default Routing Features

Normally, the firmware defaults to the equivalent behavior of the '0F' command. This combination allows for up to 4 TTL/Pulse inputs (or 2 phase-encoded counter sources), 3 routed serial boards, and 8 relays to be supervised by the code.

Any legal values for the 'F' command (as described on page 28) may be requested at the time of order. This 'default' value for the 'F' command is only used on a power-on of the board. Soft restarts (such as those caused by the '!' command or through shorting the 'RST' signal to ground) attempt to reuse the current 'F' setting, if it has not been garbled.

Default Relay Settings

Normally, the firmware defaults to a default relay setting at power on which has all of the odd-numbered relays open and the even numbered relays closed. This behavior is equivalent to the effects of the command "170R". Please refer to the 'R' command on page 32 for information about the bit assignments for this feature.

At the time of ordering the board, any legal value may be requested for this default value. Note that if the 'SR9' option is being ordered (that is to say, relay outputs are being replaced with more TTL-Serial lines), this setting is always forced to be '0R', in order to guarantee correct connections to the child boards.

SERROUTE Serial Router for GenStepper based products

The SerRoute product, when implemented on the SimStepA04 or SS0705 boards, provides for routing of serial data between one host and 3 to 9 "child" boards, each of which may be a SimStep, BiStepA04, BiStepA05, or another SerRoute board. The variance in the number of serial lines comes from the ability of the board to redefine the use of two of its three I/O ports. The package operates in one of may modes as described by the 'F' command on page 28: the most commonly used modes are:

0 - 3 Routed Serial, 8 Relay Out, 4 TTL In

1 - 5 Routed Serial, 8 Relay Out, 0 TTL in

2 - 7 Routed Serial, 0 Relay Out, 4 TTL in

3 - 9 Routed Serial, 0 Relay Out, 0 TTL in

That is to say,

TTL Input is either enabled or disabled (disabled adds 2 routed serial ports). If it is enabled, then the TTL signals may be simple TTL levels, pulses, or encoder signals from a phase encoder (version 1.17 and later of the firmware).

RELAY output is either enabled or disabled (disabled adds 4 routed serial ports or 4 encoders)

2 additional serial lines may be configured either as serial or as encoder inputs.

The routing itself is performed by inserting route commands into the serial bit stream. By default, the firmware accepts both binary-based commands (commands which contain the complete route information relative to the current board embedded within a single character) and ASCII-based commands (commands which are legible ASCII sequences, but take multiple characters to specify a route). There exists a feature option which suppresses all of the ASCII routing behavior, in case the board is used to operate non-GenStepper based products (see the 'F'eature command). When the binary-only feature is enabled, all non-null serial values less than 0xF0 can become available for both transmission and reception (if you use a router base address of 0xF0).

Routing Features

Much of the rest of this document outlines the various routing features of the code. Note that ALL of the following features are normally fully enabled in the product. However, if you set 'F'eature bit 2, only the binary modes will be enabled. Also note, that if binary routing is being used, you can minimize the "stolen character" footprint, by assigning the highest possible route number to a board. Since the boards self-configure their binary route level based on the highest character received, you can simply set your base route address to 0xF0 (240 in base 10), thus allowing all characters below 0xF0 to be transmitted on to child boards unchanged.

Direct Control Option

The direct control option of managing routing is the easiest to implement, and has few side effects. As described under the serial command sections, the "Single-level change route" command is an ASCII character, which is evaluated in the same fashion as all value/commands within the existing GenStepper series of products. That is to say, the board maintains a "most recent numeric value seen" parameter, and then interprets a single character (the "}" character) to request a "change route" action. The "}" IS sent to the current selection BEFORE the selection takes place, so that the numeric value will be terminated.

Logical "board numbers" are from 0 through 9: 0 through 8 are the targeted routings, while 9 says "broadcast to all boards, echo results to host from none". Any other value is treated internally as "access the router". Thus we have:

```
0-8     Select the request board
 9      Broadcast to all boards
<0 or >9 Select the router
```

For example,

```
"0}B200G1}X32G"
```

would be parsed as:

```
0}      Select board 0 for routing
B       On board 0, select both motors
200G    On board 0, both motors, goto location 200
1}      Select board 1 for routing (Board 1 will see "1}")
X       On board 1, select motor X
32G     On board 1, motor X, goto location 32.
```

If a SimStep or SS0705 board is the one used by the firmware, 3-9 boards (6-18 motors) can be controlled via one serial port. If any of the BiStep boards are used as the router, then 3-5 motor controller boards (6-10 motors) may be controlled using this method.

At 9600 Baud, we can only send about 960 characters/second. Assuming that we are only sending "go to" commands, with average locations comprising of 6 digits, we get 7 characters/goto. The following table shows the maximum number of GOTO commands which can be updated in 1 second using the indicated routing, assuming that each motor is separately addressed, and that both motors for a given board are sequentially addressed (thus giving us 14+selection overhead characters/motor pair)

Levels Max GOTO/second

0 (i.e., direct connection to final motor controller)

2x960/14 = 137

1 (up to 9 motor boards (18 motors), using current artwork)

2x960/(14+2) = 120

Nestable Routing Option

The nestable routing option of implementation is the most complex, from a software point of view, but has the capability of being able to control truly absurd numbers of motors. Note that, even though the control is there, given the slow communication rate specified (9600 baud, no significant buffering in the router) there is usually little real reason to go more than 2 routers deep. However, for completeness, the following is the implementation for "nestable routing".

The router has two modes of board selection (both operate simultaneously); the last one used is the one active at any given time. See the "serial commands" of "Binary Board Routing Select" and "ASCII Board Routing Select" for details of how they work.

Both of the above methods fully support a simple "tree" oriented scheme of expansion:

```
HOST --- Main Router (addressed as "{}" or "0xFF")
         |
         +-- Sub-Router 0 (addressed as "{0}", or "0xF0 0xEF")
         |    |
         |    + Motor Controller 00  (addressed as "{00}" or "0xF0 0xE0"
         |    |      + Motor X
         |    |      + Motor Y
         |    + Motor Controller 01  (addressed as "{01}" or "0xF0 0xE1"
         |    |      + Motor X
         |    |      + Motor Y
         |    ...
         +-- Sub-Router 1 (addressed as "{1}", or "0xF1 0xEF")
         |    |
         |    + Motor Controller 10  (addressed as "{10}" or "0xF1 0xE0"
         |    |      + Motor X
         |    |      + Motor Y
         |    + Motor Controller 11  (addressed as "{11}" or "0xF1 0xE1"
         |    |      + Motor X
         |    |      + Motor Y
         |    ...
         ...
```

Note that there is a limit of 8 levels of hierarchy using the binary control method, and no limit using the ASCII method.

Use of existing SimStep boards "coerced" into the new routing method gives you 1-9 "child" boards per router board. This means that, using SimStep boards, a 2-level routing address gives us up to 9x9 or 81 final low-level selected SimStep or BiStep boards. If either of the BiStep products are used, you get up to 5 "child" boards capable of being controlled, and 8 fixed relay lines (of higher power than those supported by the SimStep).

3 levels of routing using the SimStep could give us as many as 9x9x9 or 729 "real" boards (1458 motors!) being controlled by one serial line.

From a performance point of view, the multi-level nesting is cute, but not really useful beyond two levels. At 9600 Baud, we can only send about 960 characters/second. Assuming that we are only sending "go to" commands, with average locations comprising of 6 digits, we get 7 characters/goto. The following table shows the maximum number of motors which can be updated in 1 second using the indicated routing, assuming that each motor is separately addressed, and that both motors for a given board are sequentially addressed (thus giving us 14+selection overhead characters/motor pair)

Levels	**Max GOTO/second**
0	(i.e., direct connection to final motor controller) 2x960/14 = 137

1 (up to 9 motor boards (18 motors), current artwork)
2x960/(14+1) = 128 (binary method)
2x960/(14+3) = 112 (ASCII method, using "{#}")

2 (up to 9x9=81 motor boards (162 motors), current artwork)
2x960/(14+2) = 120 (binary method)
2x960/(14+4) = 106 (ASCII method, using "(##}")

3 (up to 9x9x9=729 boards (1458 motors), current artwork)
2x960/(14+3) = 112 (binary method)
2x960/(14+5) = 101 (ASCII method, using "{###}")

External Connection Summary

This section summarizes the connections used to wire the SerRoute product to your host computer and to your "child" SimStep/BiStep/SerRoute boards, as perceived by the firmware. You should also review the more complete section entitled "Board Connections" (starting on page 50) for full details of all of the connectors and their pin assignments, including schematic drawings. **Note that, in every case, any pin labeled as "Serial Input" or "Serial Output" is labeled relative to the SerRoute board**. You will connect any SerRoute "Serial Output" line to the appropriate "Serial Input" line of the "other" board. Similarly, you will connect any SerRoute "Serial Input" line to the appropriate "Serial Output" line of the "other" board.

When connecting boards together using the TTL I/O (J1 or individually labeled, depending on the board), X, and Y connectors***, there may be no RS232 line driver chips installed on the same signals***. For example, connecting a board with the SerRoute firmware to a BiStepA05 would include removal of the MAX232 chip from the BiStepA05, so that there would not be two packages driving the SimStep "Serial Input" pin B6 at the same time. Equivalently, if the child board has a 'JS' jumper as opposed to a removable MAX232 chip (as with the BiStepA06), that jumper will need to be removed for the board to act as a SerRoute child.

In all cases, the term "TTL I/O" describes a particular signal convention.

- On firmware version 1.6, voltages which are <=0.8 volts are considered to be "0", while voltages >=4.2 volts are "1". Voltages in the range of 0.9 to 4.1 are transitional, and are ignored by the processor.
- On firmware version 1.8, voltages which are <=2 volts are considered to be "0", while voltages >=3 volts are "1". Voltages in the range of 2.1 through 2.9 are transitional, and will be randomly treated as "0" or "1" by the processor.

All of the input TTL input signals are internally "pulled up" to 5 volts, using a very weak resistor. This allows the signals to be sensed as "high" if they are not connected to anything, and "low" if they are grounded; this means that, for use in the TTL input mode (the LY-, LY+, LX- and LX+), you have the option of simply using a mechanical switch-to-ground as your method of providing input to the given signal.

Labeling Of Board Signals

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

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

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

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

Release 2 – BiStepA06, BiStep2A, and SS0705

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

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

TTL and Encoder Input, or Serial Ports 3 and 4

Lines A0/LY- through A3/LX+ are used by the software to either input TTL/Encoder signals (either as standard direct output from a TTL device, or as switch closures to ground), or as an additional pair of serial ports for controlling two child boards. At power on, the system defaults to using these as serial input ports; however, through use of the "F"eatures command bit 0, these may be reassigned as serial ports numbers 3 and 4. Please review the section of this manual which describes the "F" command (page 28) for further information.

When used as serial ports, the signals are TTL-serial. That is to say, no RS232 level conversion is provided on the board.

The connections are:

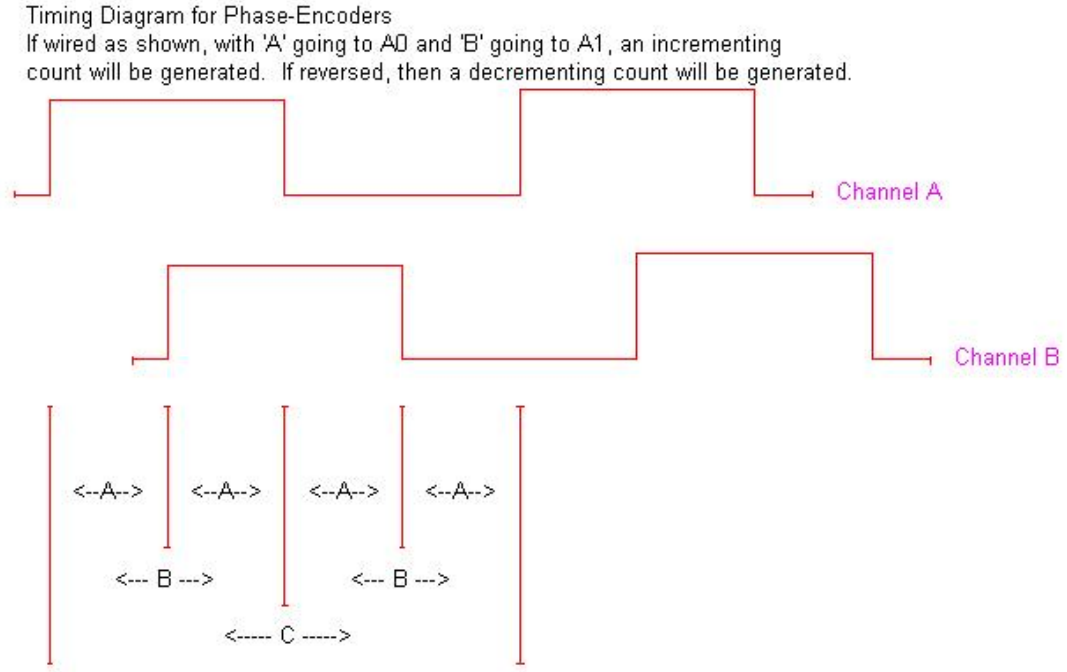
Signal	***TTL Input Data Value***	***Serial Port #***	***Serial Port Use***
A0/LY-	+1, may be optionally used as a pulse count input. In firmware versions 1.17 or later, as one side of phase-encoded counter 0	3	Serial Input
A1/LY+	+2, may be optionally used as a pulse count input in firmware versions 1.14 and later. In firmware versions 1.17 or later, may be used as the other side of phase-encoded counter 0	3	Serial Output
A2/LX-	+4, may be optionally used as a pulse count input in firmware versions 1.14 and later. In firmware versions 1.17 or later, may be used as one side of phase-encoded counter 1	4	Serial Input
A3/LX+	+8, may be optionally used as a pulse count input in firmware versions 1.14 and later. In firmware versions 1.17 or later, may be used as the other side of phase-encoded counter 1	4	Serial Output

Encoder Connections (firmware 2.0 or later)

Up to 8 phase-encoded positional encoders may be connected to most of the TTL-Serial lines, if the appropriate input mode is enabled (see page 28). The mappings areas follows: the 'Report' column describes which report command to issue in order to see the current encoder value.

Encoder	Signals	Report
0	LY-/LY+	-4?
1	LX-/LX+	-5?
2	Y-/Y+	-6?
3	X-X+	-7?
4	Y:WA1/WA2	-8?
5	Y:WB1/WB2	-9?
6	X:WA1/WA2	-10?
7	X:WB1/WB2	-11?

In each case, the code has the following signal requirements:



If the firmware senses that a pulse edge on one signal (such as channel B going low-to-high) occurs within x microseconds of a pulse edge on the other signal (such as channel A going high-to-low), then the firmware will flag that channel as having had a counter overrun. The effect of this transition from the point of view of the firmware is that both pulses will have changed within one polling cycle: since they both changed, the firmware cannot tell how to update the associated counter.

Key 'x' timing value for encoders and pulses

The actual value for the 'x' timing parameter as used as the minimum pulse separation width for encoders and as the minimum pulse width on TTL pulse inputs depends on the number of encoders which are active, and on whether the X and Y connectors are being used as serial ports (SR9 option). For firmware version 1.17, the 'x' value is 8 microseconds (with a 'P' time of 1 millisecond). For firmware versions 2.0 and later, the following table shows the timings which are used (in microseconds): use the next lower line (next larger 'x' value) if the X and Y connectors are being used as serial ports.

Number of Encoders	'X' value (minimum pulse separations)	Timing of 'P' command (125 ticks)
0	7.44 microseconds	930 microseconds
1-2	7.44 microseconds	930 microseconds
3-4	8.64 microseconds	1080 microseconds
5-6	10.40 microseconds	1300 microseconds
7-8	13.04 microseconds	1630 microseconds

Under firmware version 2.0, the current timing being used by the code is available through use of the "10?" report, combined with some simple processing. "10?" reports a byte value which is processed as follows in order to form the above microsecond reading:

x=((256 - <10? value>) * 80)/1000

I.e., the '10?' reading, when subtracted from 256 and then multiplied by 80, yields the number of nanoseconds which are the key timing parameter.

See page 33 for information on how to reset the encoders through use of the 'E' and '=' commands.

Serial Ports 0-2

Lines B0/Y- through B5/READY are normally assigned as serial ports for routed use by the system. They are all treated as TTL-serial; that is to say, no RS232 level conversions are done on the board. Firmware versions 2.0 and later support use of lines Y- through X+ as two additional encoder inputs, if so configured through the 'F' command on page 28.

The connections are:

Signal	Serial Port #	Serial Port Use	Encoder Use
B0/Y-	0	Serial Input	ENC 2 'A' line
B1/Y+	0	Serial Output	ENC 2 'B' line
B2/X-	1	Serial Input	ENC 3 'A' line
B3/X+	1	Serial Output	ENC 3 'B' line
B4/NXT	2	Serial Input	<not available>
B5/RDY	2	Serial Output	<not available>

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 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.**

X and Y connectors – 8 Relays, 4 encoders, or 4 more serial ports

The X and Y connectors may be used to operate up to 4 relays (each), 4 encoders, or 4 more serial ports. The selection as to which mode of operation is used is made by the "F"eatures command, (see the "F" command on page 28).

When used as relay control, one line from the relay is connected either to ground (if using a board from the BiStep series as the router) or to one of the +Vm 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" compatibility with serial operations. This means that the even number relays are set to 0 (open), while the odd numbered relays are set to 1 (closed).

When used as serial or encoder lines (only available on the SimStep and SS0705 boards), the ULN2803A buffer driver must be removed, and the top 8 pairs of pins "connected" together using an 8-line DIP 1K resistor pack (such as the DigiKey 4116R-1-102-ND). The bottom pair of pins (pins 9 and 10 on the socket) must be left open; otherwise, you would be shorting power to ground! Once the resistor pack is installed, then the pin assignments shown in the following table will be correct.

Signal	***Relay Bit***	***Relay Value***	***Serial Port #***	***Serial Port Use***	***Encoder Channel***	***Encoder Signal***
Y-WA1	0	+1	5	Serial Input	4	A
Y-WA2	1	+2	5	Serial Output	4	B
Y-WB1	2	+4	6	Serial Input	5	A
Y-WB2	3	+8	6	Serial Output	5	B
X-WA1	4	+16	7	Serial Input	6	A
X-WA2	5	+32	7	Serial Output	6	B
X-WB1	6	+64	8	Serial Input	7	A
X-WB2	7	+128	8	Serial Output	7	B

Summary of All Connections

The following table is a summary of all connections which are possible using the SerRoute firmware, assuming that the given board supports it:

Signal	***TTL Input Bit, Value***	***Encoder Input channel, signal***	***Relay Output Bit, Value***	***Serial Port #***	***Child Board Serial Port Connection***
B0/Y-		2, side A		0 Serial Input	Board 0 SEROUT (B7/SO)
B1/Y+		2, side B		0 Serial Output	Board 0 SERIN (B6/SI)
B2/X-		3, side A		1 Serial Input	Board 1 SEROUT (B7/SO)
B3/X+		3, side B		1 Serial Output	Board 1 SERIN (B6/SI)
B4/NX				2 Serial Input	Board 2 SEROUT (B7/SO)
B5/RDY				2 Serial Output	Board 2 SERIN (B6/SI)
B6/SERIN/SI				(Board Serial Input)	
B7/SEROUT/SO				(Board Serial Output)	
A0/LY-	0,+1	0, side A		3 Serial Input	Board 3 SEROUT (B7/SO)
A1/LY+	1,+2	0, side B		3 Serial Output	Board 3 SERIN (B6/SI)
A2/LX-	2,+4	1, side A		4 Serial Input	Board 4 SEROUT (B7/SO)
A3/LX+	3,+8	1, side B		4 Serial Output	Board 4 SERIN (B6/SI)
Y-WA1		4, side A	0,+1	5 Serial Input	Board 5 SEROUT (B7/SO)
Y-WA2		4, side B	1,+2	5 Serial Output	Board 5 SERIN (B6/SI)
Y-WB1		5, side A	2,+4	6 Serial Input	Board 6 SEROUT (B7/SO)
Y-WB2		5, side B	3,+8	6 Serial Output	Board 6 SERIN (B6/SI)
X-WA1		6, side A	4,+16	7 Serial Input	Board 7 SEROUT (B7/SO)
X-WA2		6, side B	5,+32	7 Serial Output	Board 7 SERIN (B6/SI)
X-WB1		7, side A	6,+64	8 Serial Input	Board 8 SEROUT (B7/SO)
X-WB2		7, side B	7,+128	8 Serial Output	Board 8 SERIN (B6/SI)

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 9600 baud, no parity, and 1 stop bit.

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,

5C

would mean "Close relays 0 and 2"

0R?

would mean "open all relays, then do a diagnostic summary of all current parameters".

The firmware actually recognizes and responds to each new command about ½ of the way through the stop bit of the received character. This means that the command starts being processed about ½ bit-interval before completion of the character bit stream. In most designs, this will not be a problem; however, since all commands issue an '*' upon completion, and they can also (by default) issue a <CR><LF> pair before starting, it is quite possible to start receiving data pertaining to the command before the command has been fully sent! In microprocessor, non-buffering designs (such as with the Parallax, Inc.[tm] Basic Stamp [tm] series of boards), this can be a significant issue. All firmware versions 1.4 and above handle this via a configurable option in the 'V' command. If enabled, the code will "send" a byte of no-data upon receipt of a new command character. This really means that the first data bit of a response to a command will not occur until at least 9 bit intervals after completion of transmission of the stop bit of that command (about 900 uSeconds at 9600 baud); for the Basic Stamp[tm] this is quite sufficient for it to switch from send mode to receive mode.

Software Synchronization Notes

Changes to existing software used to control the GenStepper firmware is minor; it is suggested that code which "looks" for outgoing command characters simply be modified to not expect a response when it "sees" any character > '{' being sent (0x7B). The GenStepper firmware (version 1.57 and above) now ignores such characters (aside from stopping interpretation of numeric values), so that spurious data is not sent back to the host. Otherwise, you simply insert routing commands into the stream, when you want to change control to a new motor.

You may need to adjust code which actively waits for "*" response characters to be a little more aggressive; only wait when you really need to, and do an active synch before you start doing something which is data critical. Remember that the SerRoute firmware uses the same techniques of echoing data to the host as does the rest of the product line; if it sees new data FROM the host, it stops sending data TO the host until it has processed the new character. It actively skips (drops) data until the processing starts; thus, any buffered data (such as copyright notices) will be lost. This means that, for full, clean resynch to data (such as issuing the 'i' command to a stepper board to wait for motor motion complete), you should send several numeric characters (such as '00') first (which will clear all transmissions from the board), then clear your input buffers (after the second numeric digit; you will normally no longer be receiving data here...) then send your command data and command. You can then safely expect that the complete data will be correctly received. To summarize:

1. Send "00" (actually, if you run more than 1 level deep of routing, you may want to send one extra '0' per route level)

2. Clear input buffers

3. Send command data (if any), then the command

4. All data coming back should be the requested response; '*' is your "command completed" response

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.

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 board feature set to 9-port serial mode

\ – Transmit the next (following) character unchanged (when routing): do not interpret it

As of firmware version 2.1, if you insert an '\' in the serial stream while you are routing data, then the following character is transmitted without being interpreted by the firmware. This is the only way to get Null characters or any characters from the set of "{}<>\" transmitted to the child board, and is an alternate method for allowing characters with bit 7 set (>= 0x80) to be transmitted unchanged.

>=0x80 – Binary Board Routing Select

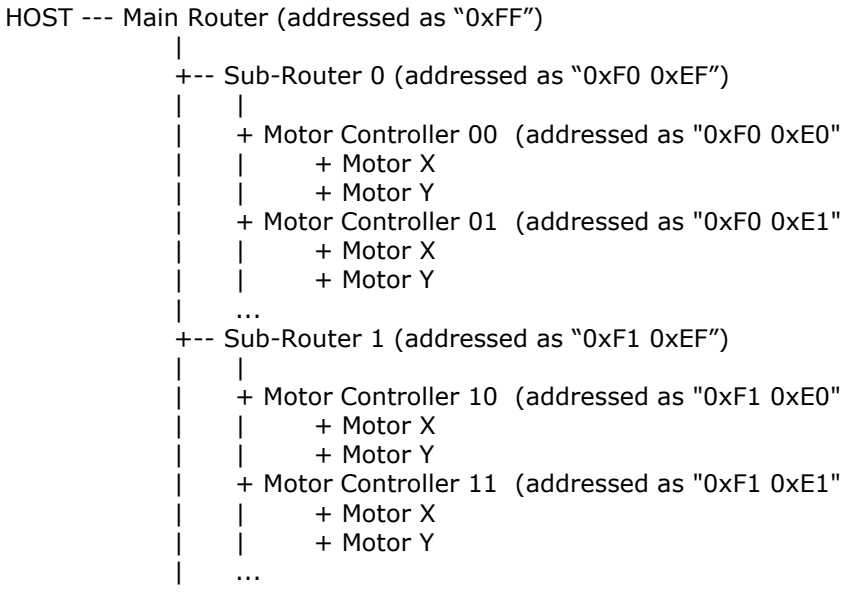
The Binary Control routing method allows a single byte to be used to select routing to any of the serial child boards, and allows for up to 8 levels of boards to be controlled (i.e., 9^8 power of final low-level boards) with an 8 byte selection method. The control is based on bit 7 being on in an input character. Please note that, as of firmware version 2.1, this feature may be disabled by setting bit 7 of the 'F'eatures command, if you would prefer to have binary data transmitted unchanged without use of the escape ("\") character.

The algorithm operates internally as follows:

1. Set "Current Command Level" to 0x80 (initialize), and current I/O board ID to board 0. Note that any character whose binary value is >=0x80 becomes a potential "board select" request character, and the "current command level" (below) is always the highest board select character processed with the low 4 bits set to 0.
2. For each input character:
   - If the new byte seen is < "Current Command Level" then send it on unchanged to current output port. This allows automatic nesting of SerRoute boards; the 'highest numbered' boards are the ones closest to the host, in the daisy-chain of wiring.
   - Else this is a new routing command: If (new seen AND 0xF0) > "Current Command Level" then reset "Current Command Level" to (new seen AND 0xF0). Note that this behavior controls the nesting of command levels. The actual serial port is set to <new seen AND 0x0F> (board 00 to 0F)

Board 09 is reserved for "broadcast to all boards"; this is used for fast actions which reference everything; no data is echoed back to the host in this mode.

Note that boards 0A-0F are for direct board commands: all initially map into the same internal routing, and enable the serial data to come to and from the board directly (thus allowing reset, data read, etc.)



{xxx} –ASCII Board Routing Select

The ASCII Readable routing method allows for the byte stream used to select the board to be controlled purely by legible ASCII characters. The intent is to produce an expandable, nestable system which generates request streams which are easy to control using simple text files.

The method is to wrap the "board selection" request in curly braces ("{}"). When a leading curly brace is seen, the board enters "select mode" for the current level. If a trailing brace is immediately seen (i.e., "{}"), then the board itself is selected (similar to sending 0xFF in the binary method). Otherwise, the characters seen are parsed to select the board. The first character after the "{" selects which board is to be selected by the current board. If the next character is NOT a "}", then the board will send a "{", followed by all remaining characters until the final "}" is seen, to the selected board (thus sending a shorter address to the next board in line, which will itself strip a character and send it on, thus allowing as long of a daisy chain as is desired). Note that the selection characters may only be digits ("0"-"9"); other characters which are not "{" or "}" are dropped by the board.

If a new "{" is received while processing an old one, OR if a binary board select is received while processing an ASCII board select, then we restart the board select rule set.

The method would therefore look as follows on parsing "{137}"

Character	Action
{	Start processing the new address: * If already processing an address, sends abort selection as needed * Stops sending of data from existing selection to host
1	Select board 1 for I/O Continue blocking data sent to host
3	Queues "{3" to board 1 (the currently selected board), Continue blocking data sent to host
7	Queues "7" to board 1 Continue blocking data sent to host
}	Queues "}" to board 1 Enable sending new data from board 1 to host

Note that, as a result of the above processing, Board 1 in the above chain would see:

"{37}",

Its board 3 would see:

"{7}",

And that one's board 7 would become active for communications.

Board '9' is reserved for "broadcast to all boards".

If the router just sees "{}", then it selects itself as the target for the following commands (i.e., the board will respond to the rest of the commands in this manual).

```
HOST --- Main Router (addressed as "{}")
              |
              +-- Sub-Router 0 (addressed as "{0}")
              |     |
              |     + Motor Controller 00  (addressed as "{00}")
              |     |       + Motor X
              |     |       + Motor Y
              |     + Motor Controller 01  (addressed as "{01}")
```

```
|     |       + Motor X
|     |       + Motor Y
|     ...
+-- Sub-Router 1 (addressed as “{1}”)
|     |
|     + Motor Controller 10  (addressed as "{10}")
|     |       + Motor X
|     |       + Motor Y
|     + Motor Controller 11  (addressed as "{11}")
|     |       + Motor X
|     |       + Motor Y
|     + Motor Controller 12  (addressed as "{11}")
|     |       + Motor X
|     |       + Motor Y
|     + Sub-Router 3  (addressed as "{13}")
|     |       |
|     |       + …
|     |       +Motor Controller 137 (addressed as “{137}”)
|     |       …
|     ...
...
```

x} – Single-level ASCII Board Routing selection

The direct control option of managing routing is the easiest to implement, and has few side effects. The "change route" command is the ASCII character "}", which is evaluated in the same fashion as all value/commands within the existing GenStepper series of products. The board maintains a "most recent numeric value seen" parameter, and then interprets a single character (the "}" character) to request a "change route" action. The "}" IS sent to the current selection BEFORE the selection takes place, so that the numeric value will be correctly terminated on the currently selected child board.

Logical "board numbers" are from 0 through 9. 0 through 8 are the targeted routings, while 9 says "broadcast to all boards, echo results to host from none". Any other value is treated internally as "access the router". Thus we have:

- 0-8: Select the request board
- 9: Broadcast to all boards
- <0 or >9: Select the router

For example,

"0}B200G1}X32G"

would be parsed as:

0}	Select board 0 for routing
B	On board 0, select both motors
200G	On board 0, both motors, goto location 200
1}	Select board 1 for routing (Board 1 will see "1}")
X	On board 1, select motor X
32G	On board 1, motor X, goto location 32.

x< – Single-level ASCII Board Routing selection for non-Peter Norberg Consulting boards

Introduced in firmware version 2.1, the '<' command is absolutely identical to the "x{" command, except that it does not transmit the "{" character to the selected child board. This makes this command useful for communicating with boards which are not produced by Peter Norberg Consulting, Inc., and hence do not know how to handle route requests.

">" – Instant stop routing

Introduced in firmware version 2.1, the ">" command instantly stops routing of data to any child board, returning control to the SerRoute board. No "}" is transmitted to the child boards; this makes this command useful for communicating with boards which are not produced by Peter Norberg Consulting, Inc., and hence do not know how to handle route requests.

xF – Set Board Features

This command allows you to reset the core behavior of the SerRoute board. Control over selection between TTL input and Serial I/O, Relay output and Serial I/O, binary-only routing modes, verbose responses to commands, and delayed responses to commands are provided.

Normally (assuming no special orders), the firmware defaults to the equivalent behavior of the '0F' command. This combination allows for up to 4 TTL/Pulse inputs (or 2 phase-encoded counter sources), 3 routed serial boards, and 8 relays to be supervised by the code.

The "F"eatures command is bit encoded as follows:

Bit	Add Value	Use
0	+1	0 = Select TTL/Pulse/Encoder Input mode 1 = Select Serial Routes 3 and 4 enabled
1	+2	0 = Select RELAY output mode 1 = Select Serial Routes 5,6,7 and 8 enabled
2	+4	0 = Allow ASCII modes of routing ("{" and "}") 1 = Only perform BINARY mode routes (>=0x80)
3	+8	0 = Enable Verbose Responses (extra CR/LF) 1 = Only do minimum responses (note: same effect as the 'V' command bit 0, with a reversal of the bit sense)
4	+16	0 = Fast command response 1 = Insert 1 character delay in command response (note: same effect as the 'V' command bit 1)
5	+32	Encoders 2 and 3 are enabled on input lines Y- through X+ (instead of serial routes 0 and 1)
6	+64	Encoders 4, 5, 6 and 7 are enabled on the X and Y motor connectors instead of RELAY outputs. If this is set when bit 1 is set (SerRoutes 5-8 enabled), then this bit (encoders) is ignored.
7	+128	Disable Binary Route mode (ignore bit 2); allows for easier transmission of binary data commands *(version 2.1 and later only)*

For example, to operate with minimum delays (no extra CR/LF characters), with all possible ports assigned as serial I/O ports, issue the command:

11F

Observe that 11 = 1+2+8, the sum of the values for the indicated bits.

Please note: if you are going to use both the 'F' and 'V' commands, then please be aware that both 'F' and 'V' control the verbose response mode and the fast response mode of the firmware. The last command issued is the one which will be obeyed with respect to those modes.

Please note the "Disable Binary Route" bit (bit number 7), which was introduced in firmware version 2.1. If this bit is set, then the single-character binary route mode of the firmware is blocked, and all data whose value is 0x80 or above will simply be routed to its destination (as opposed to being trapped and used as a route request). This permits use of route targets which require such data without excessive use of the escape character ("\").

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	Encoder 0 on LY-/LY+ has overflowed
1	+2	Encoder 1 on LX-/LX+ has overflowed
2	+4	Encoder 2 on Y-/Y+ has overflowed
3	+8	Encoder 3 on X-/X+ has overflowed
4	+16	System power-on or reset ("!") has occurred
5	+32	Encoder 4 on Y: WA has overflowed
6	+64	Encoder 5 on Y:WB has overflowed
7	+128	Encoder 6 on X:WA has overflowed
8	+256	Encoder 7 on X:WB has overflowed

For example, after initial power on,

```
L
```

Would report

```
L,16
*
```

If you then issue another L command, you would see:

```
L,0

*
```

since the first 'L' would reset its power-on reset flag.

I – Latched IO Port: Report current latches from the IO port, reset latches

The "I" latched I/O port command reports any state changes since the last call to "I" was made. After the request, the latched parts of the events report are set to 0, so that each "I" command reports only bits that have changed since the last call.

- The low 8 bits contain the raw IO port (LIM and SLEW inputs).
- The next 8 bits state which of those bits changed since the last "I" command.
- The next 8 bits state which bits went low since the last "I" command.
- The high 8 bits state which bits when high since the last "I" command.

Note that if you run the board in the mode which enables encoders 2 and 3 ("32F"), then all of the signals on the SLEW connector are available, as opposed to just the inputs on the LIM connector.

The latched events reported are as follows:

Bit	***Value***	***Description***
0	+1	LY-
1	+2	LY+
2	+4	LX-
3	+8	LX+
4	+16	Y-
5	+32	Y+
6	+64	X-
7	+128	X+
8	+256	LY- had an edge change
9	+ 512	LY+ had an edge change
10	+ 1024	LX- had an edge change
11	+ 2048	LX+ had an edge change
12	+ 4096	Y- had an edge change
13	+8192	Y+ had an edge change
14	+6384	X- had an edge change
15	+32768	X+ had an edge change
16	+65536	LY- had a low edge change
17	+131072	LY+ had a low edge change
18	+262144	LX-had a low edge change
19	+524288	LX+ had a low edge change
20	+1048576	Y- had a low edge change
21	+2097152	Y+ had a low edge change
22	+4194304	X- had a low edge change
23	+8388608	X+ had a low edge change
24	+16777216	LY- had a high edge change
25	+33554432	LY+ had a high edge change
26	+67108864	LX- had a high edge change
27	+134217728	LX+ had a high edge change
28	+268435456	Y- had a high edge change
29	+536870912	Y+ had a high edge change
30	+1073741824	X- had a high edge change
31	+/-2147483848	X+ had a high edge change

For example, after initial power on, with all signals on the LIM and SLEW connectors High,

```
I
```

Could report (since all bits went from low to high)

```
I,-16711681
*
```

If you then immediately performed another report request (and no inputs changed), you could get

```
I,255
*
```

If LY- then went low, it could then report

```
I,16711422
*
```

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.

Relay	Add Value
0	+1
1	+2
2	+4
3	+8
4	+16
5	+32
6	+64
7	+128

For example,

5C

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

xO – Open (release) selected relays

This sets the requested relay bits to '0', thus opening the indicated relay(s). The relays are assigned one per bit, as noted by the "C" command, above.

For example,

10o

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

xR – Set all 8 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 assigned one per bit, as noted by the "C" command, above.

For example

7r

will close relays 0, 1 and 2; and will open relays 3-7.

xE – Select encoder for reset by '=' command

If TTL input mode is enabled, then lines A0 through A3 (LY- through LX+) are always monitored as if they are encoder inputs. This means that all transitions on any of those lines will result in updates to the two 32 bit encoder counters. The 'E' command allows selection of which of those counters are to be reset in subsequent '=' commands. By default, all counters are selected for update after power on or reset.

Please see the general value report command '?' (starting on page 39), for information on how to retrieve counter data.

The 'x' value is bit encoded to select which encoder is to be accessed:

Bit	Value	Description
0	+1	Counter for Encoder 0 (on LY-/LY+)
1	+2	Counter for Encoder 1 (on LX-/LX+)
2	+4	Counter for Encoder 2 (on Y-/Y+)
3	+8	Counter for Encoder 3 (on X-/X+)
4	+16	Counter for Encoder 4 (on Y:WA)
5	+32	Counter for Encoder 5 (on Y:WB)
6	+64	Counter for Encoder 6 (on X:WA)
7	+128	Counter for Encoder 7 (on X:WB)

If all encoder selection bits are 0, then the code forces all encoders to be selected (i.e., an x value of '0' gets changed to an x value of '255').

For example,

```
2e
0=
1e
1000=
```

will set encoder counter 0 to 1000 and encoder counter 1 to 0.

x= – Assign current encoder value

If the appropriate mode is enabled via the 'F' command, then the requested lines are always monitored as if they are encoder inputs. This means that all transitions on any of those lines will result in updates to the two 32 bit encoder counters. The '=' command allows those counters to be reset to user specified values. It works in conjunction with the 'e' command (above), in that it assigns the requested 'x' value to the counters which have been selected through use of the 'e' command.

Please see the general value report command '?' (starting on page 39), for information on how to retrieve counter data.

For example,

```
255e
0=
```

will assign all counters a value of 0.

xP – If TTL input mode is enabled, count pulses or measure pulse widths on line A0/LY-

The 'P' command has two modes of operation. One causes the code to both set a sample period (to 'x' near-milliseconds), and to actually count the number of edges (low-to-high and high-to-low) which occur on the currently selected TTL input line (defaults to A0/LY- as input; firmware versions 1.14 and later allow you to change this selection using the 'S' command). The other mode can be used to measure the pulse widths for both the high and low parts of a repetitive pulse signal.

Versions of the firmware before 1.14 only support the pulse-count mode of operation, and only accept pulse inputs on A0/LY-. Versions 1.14 and later add support for pulse width measurement and optional selection of the pulse source to be A0 through A3 (LY- through LX+).

The code operates as follows:

1. Interpret the 'x' parameter passed.
    - If it is 0, use the prior sample period (which defaults to about 10 milliseconds on power on).
    - If it is –1, do a pulse width measurement, starting at the next high-going pulse edge.
    - If it is –2, do a pulse width measurement, starting at the next low-going pulse edge.
    - Otherwise use the value passed as the desired number of near-milliseconds to count pulse edges.
2. Clear the count and sizes of pulses seen.
3. Send a 'P' back to the host, to tell it that counting/measurement has started.
4. If pulse counting, start counting pulses, and continue counting for the specified number of near-milliseconds. If measuring pulse widths, measure the next pulse (both high and low), starting at the requested pulse edge.
5. If a new character is received before the sample period has completed, abort the request.
6. Otherwise, report as is appropriate for the request.

If requesting a pulse count (that is to say, the 'x' parameter is >= 0), then 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 "10p" to sample for 10 milliseconds, the value reported will be 20 counts (for firmware versions prior to 2.0).

For example,

10p

could report

P,20

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

0P

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

The minimum detectable pulse width is x microseconds, where 'x' is derived based on the configuration of the board. Also, the real sample period is in units of 125*x, which is usually in the neighborhood of 1 millisecond. Please see page 17 for information on how to derive the 'x' value, and for the exact sample period values used. For example, if 'x' were 8 microseconds (firmwares prior to 2.0), this would mean that the code could not detect

more than 125,000 edges (1 million divided by x) per second; and that input frequencies which exceed 62,500 (500,000 divided by x) Hz would not be correctly interpreted.

If requesting a pulse width measurement (available in version 1.14 and later of the firmware), then the system scans for the requested leading edge (low going or high going), and then counts both the low and high times. The report includes both values.

"-1P" Report starts at High going edge

"-2P" Report starts at Low going edge

In all cases, the report result is low, high width, in x microsecond units. If a 0 count is reported, then an overflow occurred in time; the pair should not be treated as valid.

For example, the resulting report of:

P,25,37

would mean that the low part was 25*x microseconds wide, while the high part was 37*x microseconds wide. The widths are all going to be +/- 1, since the code only samples on x microsecond intervals, and is thus asynchronous with respect to the real pulses.

The resulting report of:

P, 65423,0

would mean that the low part was 65,423 * x microseconds wide, while the high part timed out.

Note that if a 'leading edge' times out, the other half of the square wave will not be tested. Thus, if a first-half times out, you will always get a report of:

P,0,0

For example, the above "P,65423,0" report could only happen on a "-2P" request, while a "P,0,65311" report could only happen on a "-1P" request.

xS – Select Pulse Source Input Line

For firmware versions 1.14 and later, you can select the TTL input line which is used to provide the pulses for pulse count and width measurements. By default, at power on, line A0 (the LY- signal) is selected; however, any one of the four limit inputs may be selected as the source of data for the next 'P' command through use of the 'S' command.

The value provided is actually the mask used against the input port to detect changes and signal levels. This means that if you do not use one of the values listed in the following table, you will get rather strange behavior out of the firmware. Please only use the following values on the 'S' command.

Value	Pulse Source
1	A0/LY-
2	A1/LY+
4	A2/LX-
8	A3/LX+

For example, to measure the width of repetitive pulses which are appearing on line LY+, you could issue the command sequence:

```
2S
-1P
```

T – Report all TTL input values

If the 4 TTL inputs are enabled, this gives you access to the current (noise-filtered) TTL input values. As with the relays (see the prior page), the value reported is bit encoded; a "1" means that the TTL input is high, a "0" means that the associated input is low. The controller reports the sum of all of the currently selected inputs.

For example

T

could report

T,9

*

which would mean that TTL inputs 0 and 3 are high (not grounded), 1 and 2 are low (grounded).

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 1.4 and above handle this via a configurable option in the 'V' command. If enabled, the code will "send" a byte of no-data upon receipt of a new command character. This really means that the first data bit of a response to a command will not occur until at least 9 bit intervals after completion of transmission of the stop bit of that command (about 900 uSeconds at 9600 baud); for the Basic Stamp[tm] this is quite sufficient for it to switch from send mode to receive mode.

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

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

Please note: if you are going to use both the 'F' and 'V' commands, then please be aware that both 'F' and 'V' control the verbose response mode and the fast response mode of the firmware. The last command issued is the one which will be obeyed with respect to those modes.

! – RESET – Reset relays to power on default, resynch all serial

This command acts similar to a power-on reset; however, it remembers all of the currently selected "F"eatures of the board, so that a new features command is not required. All relays get reset to their power on defaults (even number open, odd number closed), and all serial ports get reset (any pending characters get lost).

For example,

!

resets the system to its currently selected feature set (the 'F' command).

x? – Report status

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

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

1-255: Report memory location 1-255

Useful locations: ***NOTE THAT ANY LOCATION ABOVE 7 MAY CHANGE BETWEEN CODE VERSIONS***

- 5: Port A register – this controls the TTL input/2 additional serial ports
- 6: Port B register – this controls the fixed serial ports
- 7: Port C register – this controls the relay/4 additional serial ports
- 10: (Version 2.0 and later): Minimum time to detect any pulse edges (encoders or TTL pulses of any form): Please see page 17 for more information on how to use this parameter.

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 relay settings (bit; 0=open, 1=closed)
- -2; Report current TTL input port data
- -3; Report current current board features (the "F" command)
- -4; Report encoder 0 counter value
- -5; Report encoder 1 counter value
- -6; Report encoder 2 counter value
- -7; Report encoder 3 counter value
- -8; Report encoder 4 counter value
- -9; Report encoder 5 counter value
- -10; Report encoder 6 counter value
- -11; Report encoder 7 counter value
- -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. A "S" character is sent, to signify that the report is from the SerRoute board.
3. A comma is sent.
4. The report number is sent (such as –1, for relay settings).
5. Another comma is sent.
6. The requested value is reported.
7. If Verbose Mode is on, then a <crlf> is sent.
8. An "*" is sent, to notify the caller that the report is complete.

The special reports which are understood are as follows:

0: Report all reportable items

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

S,0,a,b,c,d,e,f,g,h,i,j,k

Where:

- S identifies the report as SerRoute generated
- 0 is the report number; 0 is the 'all' report
- a is the value for the current relay settings (report "-1")
- b is the value for the current TTL input port data (report "-2")
- c is the value for the current feature selections (report "-3")
- d is the value for encoder 0 (on LY-/LY+, report "-4")
- e is the value for encoder 1 (on LX-/LX+, report "-5")
- f is the value for encoder 2 (on Y-/Y+, report "-6")
- g is the value for encoder 3 (on X-/X+, report "-7")
- h is the value for encoder 4 (on Y-WA1/WA2, report "-8")
- i is the value for encoder 5 (on Y-WB1/WB2, report "-9")
- j is the value for encoder 6 (on X-WA1/WA2, report "-10")
- k is the value for encoder 7 (on X-WB1/WB2, report "-11")

For example,

```
0?
```

Would report all reportable values for SerRoute. You could receive:

```
S,0,170,15,0,2500,5682734,0,0,0,0,0,0
*
```

which would mean:

```
170: Relays 1, 3, 5 and 7 closed; relays 2, 4, 6, 8 open
15:  All TTL inputs open (or high)
0:   Default feature set (TTL and RELAYS enabled…)
2500: Encoder 0 shows location of 2,500
5682734: Encoder 1 shows location 5,682,734
```

-1: Report current relay settings

This reports the current (instantaneous) relay settings. The reported value is the sum of all closed relays, per the table described in the "C" command.

For example,

```
-1?
S,-1,170
*
```

-2: Report current TTL input port data

This reports the current (noise-filtered) TTL input data (the same values as those presented by the "T" command).

For example,

```
-2?
S,-2,15
*
```

-3: Report current feature selections

This reports the current "F"eature selections.

For example,

```
-3?
S,-3,0
*
```

-4: Report encoder 0 value

This reports the current value for encoder 0 (on LY-/LY+).

For example,

```
-4?
S,-4,1000
*
```

-5: Report encoder 1 value

This reports the current value for encoder 1 (on LX-/LX+).

For example,

```
-5?
S,-5,1000
*
```

-6: Report encoder 2 value

This reports the current value for encoder 2 (on Y-/Y+).

For example,

```
-6?
S,-6,1000
*
```

-7: Report encoder 3 value

This reports the current value for encoder 3 (on X-/X+).

For example,

```
-7?
S,-7,1000
*
```

-8: Report encoder 4 value

This reports the current value for encoder 4 (on Y:WA1/WA2).

For example,

```
-8?
S,-8,1000
*
```

-9: Report encoder 5 value

This reports the current value for encoder 5 (on Y:WB1/WB2).

For example,

```
-9?
S,-9,1000
*
```

-10: Report encoder 6 value

This reports the current value for encoder 6 (on X:WA1/WA2).

For example,

```
-10?
S,-10,1000
*
```

-11: Report encoder 7 value

This reports the current value for encoder 7 (on X:WB1/WB2).

For example,

```
-11?
S,-11,1000
*
```

-12: Report current software version and copyright

This reports the software version and copyright.

For example,

```
-12?
```

could report:

```
SerRoute.src $version: 1.14$
Copyrights 2002-2004 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,

12 5r

would actually request a "Set relays to 5". Since the " " command is illegal, it is ignored; however, it terminates interpretation of the number which had been started as 12.

If verbose mode is enabled, then upon starting processing of any command (including the 'ignored' commands), the board sends the <carriage return><line feed> pair.

Note that, upon completion of ANY command (including the 'ignored' commands), the board sends the "*" character.

Basic Stamp™ Sample Code

The SerRoute series of boards may all be used with the Parallax, Inc.[tm] Basic Stamp [tm] series of boards. The connection to the SerRoute product is performed via two of the pins on the J1 connector (SERIN (B6), and SEROUT (B7)), with the MAX232 IC removed from its socket. The remaining input pins on the J1 set of connectors may be wired or not, as needed by the application. Most of the time, they will be connected to other StepperBoard boards (such as SimStep and BiStep products).

Communications between the Basic Stamp and SerRoute boards is performed 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.

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

- SERIN (B6/SI) connected to P2
- SEROUT (B7/SO) connected to P1

"Genseekserroute.bs2" is a fairly comprehensive example, in terms of showing the capabilities of the SerRoute and StepperBoard systems. It operates assuming that 2 StepperBoard boards are being controlled by the SerRoute board, both of which are operating at the full level of microstep possible (1/64 of a full step). Each motor of a given board is operated at a different speed. X is set to a maximum rate of 4000 microsteps/second (which is 4000/64 or 62.5 full steps/second), with a matching ramp rate of 4000 microsteps/second/second. Y is set to a maximum rate of 8000 microsteps/second (which is 125 full steps/second), with a ramp rate of 7000 microsteps/second/second. It also sets the "automatic full-power" step rate to be 6000 microsteps/second. Given that only Y will exceed this rate, the Y motor will switch from what ever mode it is using to full power mode during any seek which goes far enough for it to exceed the 6000 microsteps/second rate. Having gone through this setup, the loop operates similarly to that in "Gendemoser.bs2", except that the locations cycled are +16,000 and 0. If you use this demo with two identical motors, you should be able to "hear" the difference in the stepping modes, and you should also hear the Y motor "become noisy" partway through the microstep phase of the entire sequence (when it switches between microstep mode and full power full step mode).

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

Listing for GENSEEKSERROUTE.BS2

```
' ****************************************************************************
' $modname: genseekserroute.bs2$
' $nokeywords$
' Demonstates some of the serial commands using seek and serial
' response to the SimStep and BiStep
' set of controllers routed via a SerRoute unit from Peter Norberg Consulting, Inc.
'
' The tool first initializes each stepper to operate as follows:
'   64 microsteps/full step,
'   start/stop rate being 320 uSteps/second
'   ramp rate at 4000 uSteps/sec/sec for the X motor,
`               7000 uSteps/second for the Y motor.
'   Auto-power switch mode (the 'A' command) is reset to 6000 uSteps/second
'   Target ramp rate is 4000 uSteps/second for X, 8000 uSteps/second for Y
'
' This combination means that the X motor will peak at 1/2 the speed of the Y motor,
' and that the Y motor will switch to full-step full power mode during the
' midpoint of the seek.
' During the microstep pass test (when idMicroStep = 3),
' you will notice that the Y motor
' will start quietly, and then suddenly become noisy for a short period,
' and then it will quiet down again.
' This is occuring when the stepping mode switches from micro to full when
' the motor speed is faster than about 6000 uSteps per second.
'
' The system is to be configured as:
'
'   Basic Stamp:
'       P1 connected to SerRoute J1-B7/SEROUT
'       P2 connected to SerRoute J1-B6/SERIN
'
'   SerRoute:
'       Remove the MAX232 serial buffer (since above is a direct
'       connection to the Basic Stamp)
'       Then, in addition to the above two wires,
'
'           SerRoute J1-B0/Y- (Serial input, port 0)
'                connected to SimStep/BiStep board 0, J1-B7/SEROUT
'           SerRoute J1-B1/Y+ (Serial output, port 0)
'                connected to SimStep/BiStep board 0, J1-B6/SERIN
'
'           SerRoute J1-B2/X- (Serial input, port 1)
'                connected to SimStep/BiStep board 1, J1-B7/SEROUT
'           SerRoute J1-B3/X+ (Serial output, port 1)
'                connected to SimStep/BiStep board 1, J1-B6/SERIN
'
'   Then two SimStep or BiStep (or one of each...) boards, connected as shown above.
'
' Note that all motors are selected for the seek actions.
' It then enters the speed test loop.
'
' The code first waits for the stepper unit to report idle.
' and it is instructed to move +16000 (in) 1/64th steps.
' (Note that this is full step delta 125).
' This is then followed by a move to location -16000, and then a new stepping mode
' is selected.  A 1/5th second pause is inserted to make it easy to identify
' when the cycle is occuring.  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)
'
'
' ****************************************************************************

' {$STAMP BS2}


' SerRoute connected as follows
'   Serial Input P1 to SerRoute B7 Serial output
```

```
'   Serial Output p2 to SerRoute B6 Serial Input
' Then two more SimStep or BiStep boards,
'   connected to the SerRoute logical ports 0 and 1.

PortStepperSerFrom  con  1      ' Serial from stepper port
PortStepperSerTo    con  2      ' Serial to stepper port
PortStepperBaud     con 84      ' Baud rate to generate 9600 baud

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

' Code restarts here if RESET button pressed

        pause 250
           ' Wait for stepper power on cycle

        serout PortStepperSerTo,PortStepperBaud,["{}2v"]
           ' Set SerRoute to use short responses, and extended response time

        for idBoard=0 to 1
            Gosub InitBoard                                 ' Init that board
            next                                            ' And go to the next one

        idMicroStep = 0                                     ' Start at microstep 0

loop:
        For idBoard = 0 to 1                                ' For each board
          gosub SelectBoard
                  ' First, select the board
           serout PortStepperSerTo,PortStepperBaud,[dec idMicroStep,"o"]
                  ' Set microstep mode

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

           gosub WaitReady                                  ' Wait until ready

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

           gosub WaitReady                                  ' Wait until ready
           Next
        idMicroStep = (idMicroStep + 1) & 3 ' Cycle step type
        goto        loop                    ' Cycle forever

'*********************************************************************
' InitBoard: Given idBoard identifies the board to access,
' we initialize the selected stepper board as routed
'*********************************************************************

InitBoard:
        gosub SelectBoard                                   ' First, select the board
        serout PortStepperSerTo,PortStepperBaud,["1!"]
                  ' Reset the stepper board 0, set 1/64 full-step step size
        pause 1000
                  ' Wait for stepper to send its wake-up copyright text
        serout PortStepperSerTo,PortStepperBaud,["2V"]
                  ' Turn off verbose responses, add delay before response
        serout PortStepperSerTo,PortStepperBaud,["320K"]
                  ' Set Stop OK to 'can start/stop at 320 microsteps/sec'
        serout PortStepperSerTo,PortStepperBaud,["6000A"]
                  ' Set auto-switch to full power mode to 6000 microsteps/sec;
                  ' only Y will do it
        serout PortStepperSerTo,PortStepperBaud,["X"]
                  ' For demo purposes, Select just X for a moment
        serout PortStepperSerTo,PortStepperBaud,["4000p"]
                  ' For demo purposes, set X slow ramp of 4000 microsteps/sec
        serout PortStepperSerTo,PortStepperBaud,["4000R"]
                  ' For demo purposes, set X target rate of 4000 microsteps/sec
        serout PortStepperSerTo,PortStepperBaud,["Y"]
                  ' For demo purposes, Select just Y for a moment
        serout PortStepperSerTo,PortStepperBaud,["7000p"]
                  ' For demo purposes, set Y faster ramp of 7000 microsteps/sec
        serout PortStepperSerTo,PortStepperBaud,["8000R"]
```

```
                ' For demo purposes, set Y target rate of 8000 microsteps/sec
        serout PortStepperSerTo,PortStepperBaud,["B"]
                ' For demo purposes, Select both X and Y for remaining actions
        return

'*********************************************************************
' SelectBoard: Using the global value idBoard,
' select which board we want to access
'*********************************************************************
SelectBoard:
        serout PortStepperSerTo,PortStepperBaud,["{", DEC idBoard, "}"]
           ' Set the router to the requested board
        return

'*********************************************************************
' WaitReady: Issue the "I"dle command, to wait for completion of
' motion on the currently selected board
'*********************************************************************
WaitReady:
'        DEBUG "Waiting..."
        serout PortStepperSerTo,PortStepperBaud,["00I"]
           ' wait for ready; the leading 0's flush BiStep's output queue

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

SerTest.exe – Command line control of stepper motors

The SerTest.exe application (provided as part of the sample software) is a simple tool which allows command line based control of the 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!x1000g

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

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

For example,

Sertest {0}4!x1000gy-2000gi

Would:

1. Operate at 9600 baud on COM1 using a 1 minute time out
2. Select board 0 from the first SerRoute board on the serial line
3. reset the newly selected board to operate with a microstep size of 4/64
4. tell the X motor to go to location 1000,

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 /b9600 /i10000 {32}y+5000s

Would:

1. Operate using port COM2 at 9600 baud, with a timeout of 10 seconds
2. Select board 3 off of the closest serial router, board 2 off of that serial router, which is then assumed to be a SimStep or BiStep stepper controller.
3. 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.

Board Connections – Please see the "UniversalStepper" Manual

Please review the "Board Connections" section of the "UniversalStepper" manual specific to your product . This will contain the standard descriptions of the connectors and signal names.

This manual will refer to the connector names found in the UniversalStepper manual for the SimStep, BiStepA04, and BiStepA05 manuals. To match signals with other products (such as the SS0705, BiStepA06, and BiStep2A), go based on the signal name as shown on the board artwork.

TTL Input, or Serial Ports 3 and 4

Lines A0/LY- through A3/LX+ are used by the software as either input TTL input signals (either as standard direct output from a TTL/encoder device, or as switch closures to ground), or as an additional pair of serial ports for controlling two child boards. At power on, the system defaults to using these as serial input ports; however, through use of the "F"eatures command bit 0, these may be reassigned as serial ports numbers 3 and 4.

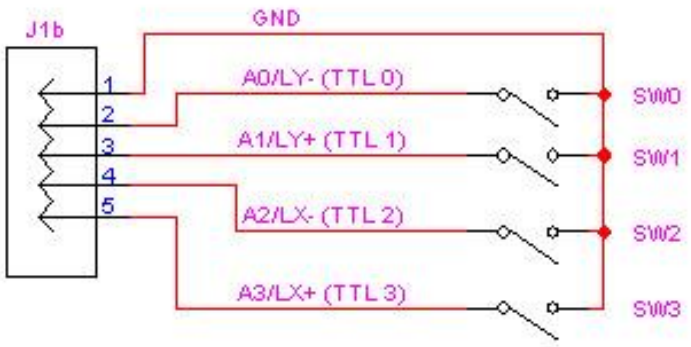
When used as serial ports, the signals are TTL-serial. That is to say, no RS232 level conversion is provided on the board.

When used as TTL input bits, the "T" command may be used to read the current values. The lines are pulled up to +5 volts via weak pull-up resistors (10-20K), and thus may be controlled via mechanical switch closures to ground, as well as via normal TTL outputs.

The connections are:

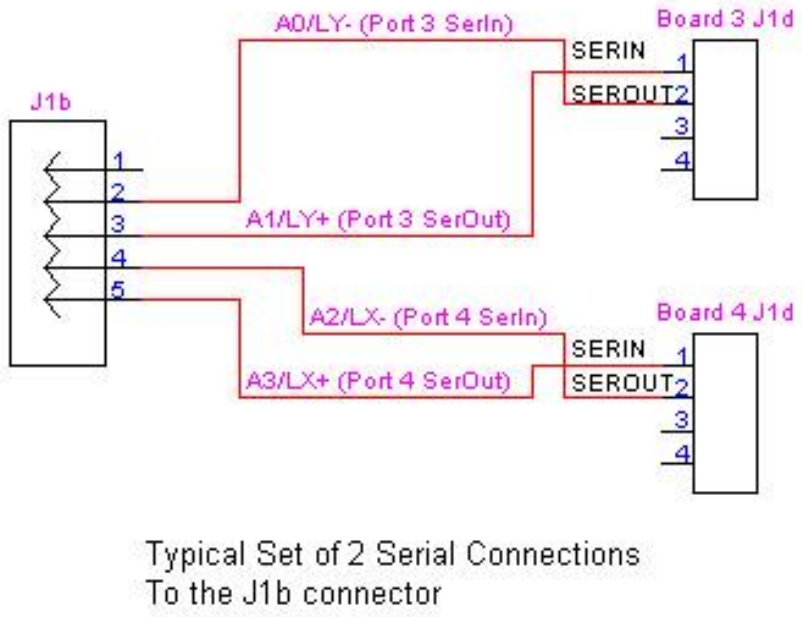
Signal	***TTL Input Bit, Value***	***Encoder Input channel, signal***	***Relay Output Bit, Value***	***Serial Port #***	***Child Board Serial Port Connection***
A0/LY-	0,+1	0, side A		3 Serial Input	Board 3 SEROUT (B7/SO)
A1/LY+	1,+2	0, side B		3 Serial Output	Board 3 SERIN (B6/SI)
A2/LX-	2,+4	1, side A		4 Serial Input	Board 4 SEROUT (B7/SO)
A3/LX+	3,+8	1, side B		4 Serial Output	Board 4 SERIN (B6/SI)

When connecting to these pins for TTL input, we could have a schematic similar to the following (the label 'J1b' means either the above signal set on the large 19 pin connector on the Revision 1 boards, or the 'LIM' connector on the Revision 2 boards):



Typical Set of 4 TTL Switch Input Connections
To the J1b connector

When connecting as two serial ports, the schematic appears as:



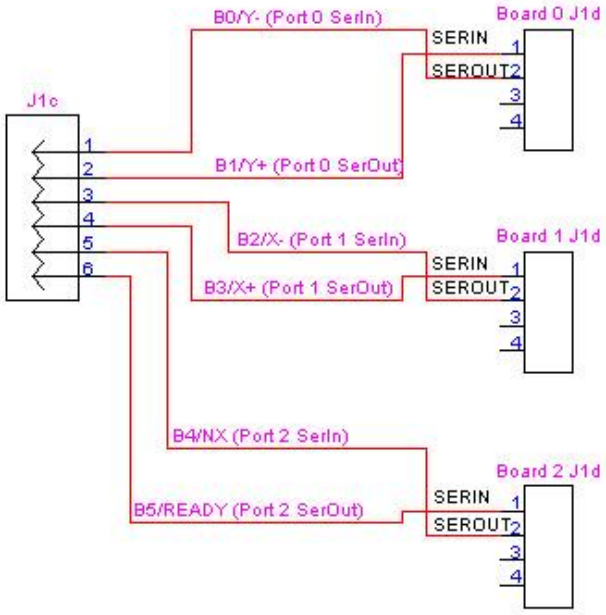
Typical Set of 2 Serial Connections
To the J1b connector

Note that the Serial Input and Output connections are always "switched"; that is to say, the serial input from one board is connected to the serial output of the other board. **The child boards (Boards 3 and 4) must have their MAX232 serial buffer chips (or their equivalent "JS" jumpers) removed**. Note also that we have labeled the SERIN/SEROUT lines as being on connector "J1d". This is either the bottom of the 19-pin J1 connector on the Revision 1 boards, or the "IO" connector on the Revision 2 boards.

Serial Ports 0-2

Signal	***TTL Input Bit, Value***	***Encoder Input channel, signal***	***Relay Output Bit, Value***	***Serial Port #***	***Child Board Serial Port Connection***
B0/Y-		2, side A		0 Serial Input	Board 0 SEROUT (B7/SO)
B1/Y+		2, side B		0 Serial Output	Board 0 SERIN (B6/SI)
B2/X-		3, side A		1 Serial Input	Board 1 SEROUT (B7/SO)
B3/X+		3, side B		1 Serial Output	Board 1 SERIN (B6/SI)
B4/NX				2 Serial Input	Board 2 SEROUT (B7/SO)
B5/RDY				2 Serial Output	Board 2 SERIN (B6/SI)
B6/SERIN/SI				(Board Serial Input)	
B7/SEROUT/SO				(Board Serial Output)	

When configured as serial, the serial ports for boards 0 through 2 are TTL-serial (that is to say, no RS232 level conversions are done on the board). Serial port 2 (NX/RDY) is always configured as a TTL-serial device. The typical schematic for use of this set of signals as TTL serial is as follows (the label J1c means either the middle of the large 19 pin connector on the Revision 1 boards, or the SLEW connector on the Revision 2 boards):



Typical Set of 3 Serial Connections
To the J1c connector

Note that the Serial Input and Output connections are always "switched"; that is to say, the serial input from one board is connected to the serial output of the other board. **The child boards (Boards 0, 1 and 2) must have their MAX232 serial buffer chips (or their equivalent jumpers) removed.** Note also that we have labeled the SERIN/SEROUT lines as being on connector "J1d". This is either the bottom of the 19-pin J1 connector on the Revision 1 boards, or the "IO" connector on the Revision 2 boards.

Raw serial I/O

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

SERIN (B6/SI) and SEROUT (B7/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 and use these pins.

Note that if this board is a "child" board in a SerRoute controlled tree of boards, then the MAX232 chip or its equivalent 'JS" jumper (as in the BiStepA06 unit) will normally be removed, unless special interfacing is done.

X and Y Relay Connectors, Wiring the Relays, additional serial ports or encoders

The X and Y connectors are used to operate up to 4 relays (each), 4 more serial ports, or 4 more encoders. The selection as to which mode of operation is used is made by the "F"eatures command on page 28.

When used as relay control, 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.

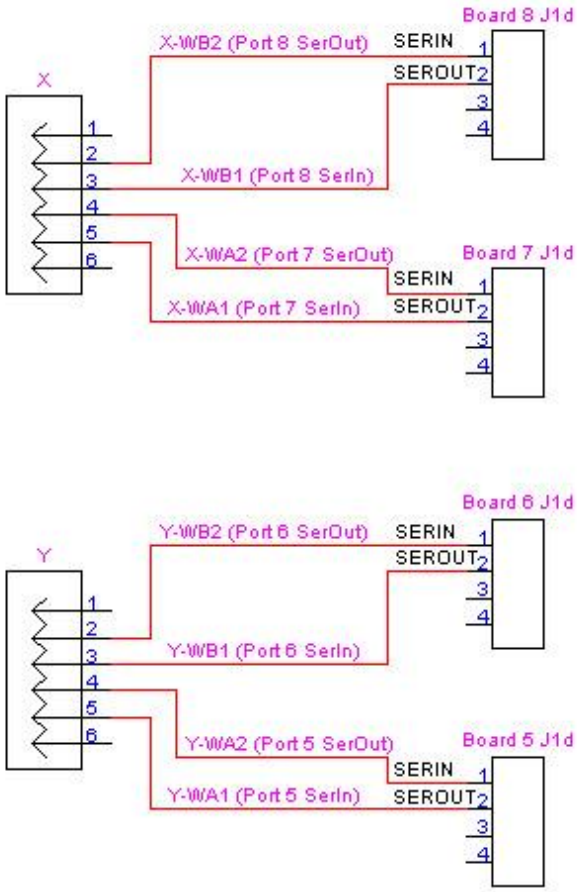
At power on and after any reset, the relays are configured for "safe" compatibility with serial operations. This means that the even number relays are set to 0 (open), while the odd numbered relays are set to 1 (closed).

The complete set of potential signal assigments for the X and Y connectors is:

Signal	***TTL Input Bit, Value***	***Encoder Input channel, signal***	***Relay Output Bit, Value***	***Serial Port #***	***Child Board Serial Port Connection***
Y-WA1		4, side A	0,+1	5 Serial Input	Board 5 SEROUT (B7/SO)
Y-WA2		4, side B	1,+2	5 Serial Output	Board 5 SERIN (B6/SI)
Y-WB1		5, side A	2,+4	6 Serial Input	Board 6 SEROUT (B7/SO)
Y-WB2		5, side B	3,+8	6 Serial Output	Board 6 SERIN (B6/SI)
X-WA1		6, side A	4,+16	7 Serial Input	Board 7 SEROUT (B7/SO)
X-WA2		6, side B	5,+32	7 Serial Output	Board 7 SERIN (B6/SI)
X-WB1		7, side A	6,+64	8 Serial Input	Board 8 SEROUT (B7/SO)
X-WB2		7, side B	7,+128	8 Serial Output	Board 8 SERIN (B6/SI)

X, Y Connectors Wired as Serial Ports

When used as serial lines (only available on the SimStep board), the ULN2803A buffer driver must be removed, and the top 8 pairs of pins "shunted" together using an 8-line DIP shunt (such as the DigiKey A26238-ND). The bottom pair of pins (pins 9 and 10 on the socket) must be left open; otherwise, you would be shorting power to ground! Once the shunt is installed, then the pin assignments shown in the above will be correct. You will need to set the "F"eature bit 1 to 1, to enable this mode of operation. For example, sending "{}2F" would enable this in a single-SerRoute system. See the section entitled "xF – Set Board Features" for more information and other options.



Typical Set of 4 Serial Connections
To the SimStepA04

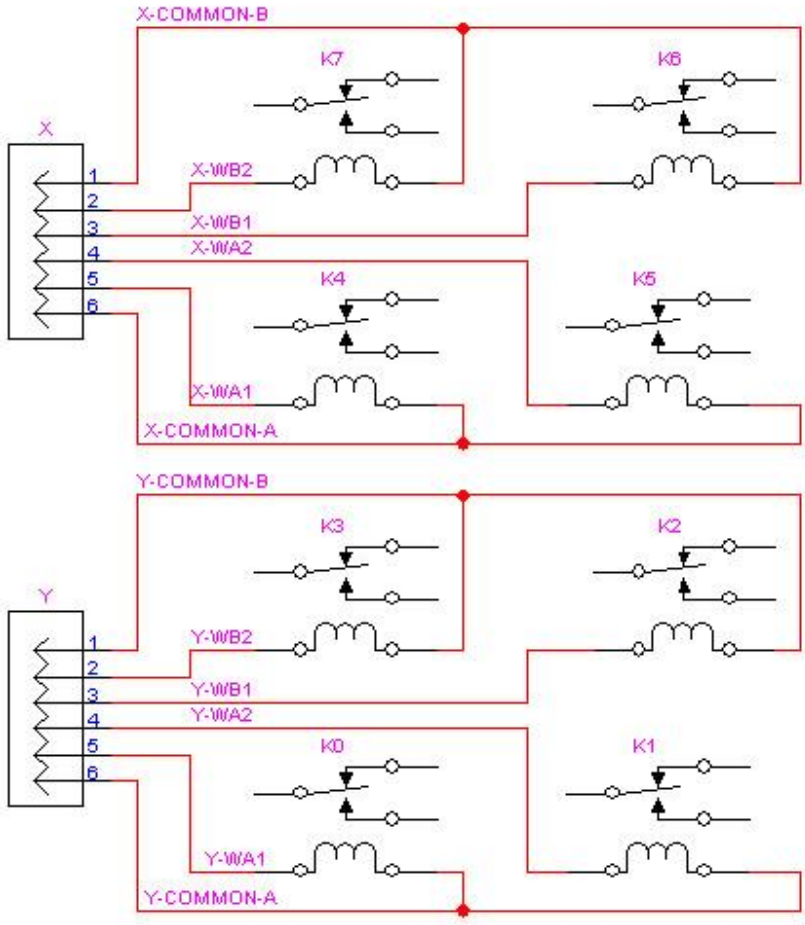
Note that the Serial Input and Output connections are always "switched"; that is to say, the serial input from one board is connected to the serial output of the other board. **The child boards (Boards 5, 6, 7 and 8) must have their MAX232 serial buffer chips (or the equivalent "JS" jumper) removed.** Note also that we have labeled the SERIN/SEROUT lines as being on connector "J1d". This is either the bottom of the 19-pin J1 connector on the Revision 1 boards, or the "IO" connector on the Revision 2 boards.

X, Y Connectors Wired As Relay Control

These two identical connectors are also used to operate the X and Y relay sets (when that mode is enabled by the firmware), respectively. They are wired as follows for the SimStepA04 and BiStep A05 (pins counting from top to bottom).

Pin	Name
1	GND or +V
2	WB2
3	WB1
4	WA2
5	WA1
6	GND or +V

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

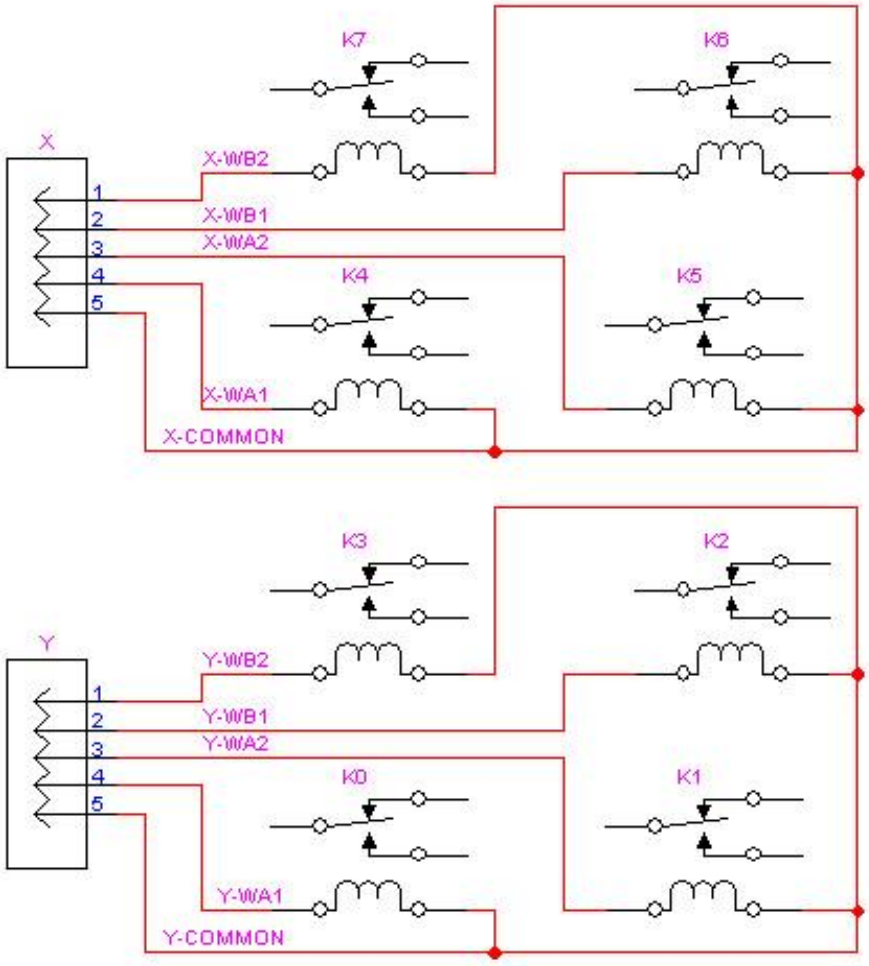


Typical Set of 8 Relay Connections
To the BiStepA05 or SimStepA04

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 8 Relay Connections
To the BiStepA04