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 NCStepper4D Firmware and the SS4D series of Stepper Motor Controllers

By

Peter Norberg Consulting, Inc.

Matches NCStepper4D Firmware Revision 1.19

Authored in the United States of America. Manual published August 28, 2008 2:30 PM

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 description. Errors may appear in the documentation; we will correct any mistakes as soon as they are discovered, and will post the corrections on the web site in a timely manner. Please refer to the specific manual for the version of the hardware and firmware that you have for the most accurate information for your product.

This manual describes artwork SS4D, using either the 0.5 or the 1.0 amp assembly option. The firmware release described is NCStepper4D version 1.19. The manual version shown on the front page normally has the same value as the associated NCStepper4D version. If no manual has yet been published which matches a given firmware level, then the update is purely one of internal details; no new command features will have been added.

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

Version	Date	Description
1.19	July 19, 2007	First manual release

Product Safety Warnings

The SS4D series of controllers 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 the board or its installed components, until the board has cooled down completely. 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 significant heat. There exist several components on the bottom side of the board that can get quite hot; therefore, the board must be correctly mounted using stand-offs.

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.

Always use a correctly grounded power supply to power the system. **Failure to do so may cause dangerous voltages to exist on the board, and thus may cause damage or injury to anything connected to the product, including people!**

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 that can cause any form of personal injury if a failure occurred.

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

Introduction and Product Summary

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

The **SS4D** series of microstepping motor controllers from Peter Norberg Consulting, Inc., has the following general performance specifications:

	SS4D05	**SS4D10**
Unipolar Motor	Yes	Yes
Bipolar Motor	No	No
Maximum Motor supply voltage (Vm)	26V	26V
Logic supply voltage (+5V)	5V	5V
Quiescent current (all windings off, no fan)	150 mA	150 mA
Maximum winding current (per motor winding, requires external fan to operate, limited duration)	0.5A	1.0A
Board size	3.70" x 1.90"	3.70" x 1.90"
Dual power supply capable	Yes	Yes
USB serial	Yes	Yes
Available in ROHS compliant version	Yes	Yes

Each board can be controlled by its 9600 baud (other rates are optionally available) serial interface, which provides full access to the controller's extreme range of stepping rates (1 to 38.422 microsteps per second), slope rates (1 to 38.422 microsteps per second per second), and various motor motion rules are provided. The boards have a theoretical microstep resolution of 1/16 of a full step, and use a constant-torque algorithm when operating in microstep mode.

The NCStepper4D firmware shares many of the features of the PotStepper4D four-motor controller firmware. The internal control of the motor drivers is identical, as is the general method of sending numeric parameters for commands. Many of the commands which configure the system are also identical (such as setting the step rate); however, the fundamental control theory is different. The NCStepper4D firmware explicitly controls all four motors at the same time, from a single command (such as Goto or Arc), with automatic step-rate ratioing in order to generate straight lines; while PotStepper4D explicitly controls the motors independently, so that one motor may be performing a "slew" operation, while another is executing a "goto".

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

```
*0X      - Set X and
*0Y      - Y center point for the arc and as the arc-based motors
*-2000R  - Set non-trapezoidal mode, so we get a fast circle
*1D      - Set the Arc-Delta to 1 degroid (1 degroid is 1/256 of a full circle; or 360/256 degrees)
*253C    - Tell the system that there will be 253 steps to draw (253 small lines)
*3c      - 3 lines as slow-down/stop, so we won't break gears
*0B      - Begin "degroid angle" is 0
*1000A   - Radius of Arc is 1000, and draw. This draws a "circle" of diameter 2000 steps.
*0X      - Reset center back to 0,0 (otherwise, new center would be the last point drawn)
*0Y
*-2000R  - Set non-trapezoidal mode, so we get a fast circle
*253C    - Reset count to 253; it gets destroyed with each draw
*3c      - 3 lines as slow-down/stop, so we won't break gears
*0B      - Reset arc angle; it is left at last point drawn
*2000A   - Draw a 2000 unit radius circle
*0X      - Once again, go to location 0,0 as center
*0Y
*4C      - This time, just set 4 lines in "arc"
*0B      - Again, start at 0 "degroids"
*64D     - set the unit delta to be 64; so that 4 will be a complete "circle"
*2000R   - Set trapezoidal mode, so we get a safe square
*3000A   - Draw the 3000 unit radius arc; since done in 4 steps, it is really a square!
*
```

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

Short Feature Summary

- Up to four stepper motors may be controlled at one time via step-and-direction pulses.
- Supports a separate backlash setting for each motor
- Limit switches may optionally be used to automatically request motion stop of any motor in either direction.
- Rates of 1 to 38,422 steps per second are supported.
- Step rates are changed by linearly ramping the rates. The rate of change is independently programmed for each motor, and can be from 1 to 38,422 steps per second per second.
- All motor coordinates and rates are always expressed in logical step units. The actual step sizes that are used are dependent upon the current 'microstep size' programmed by you into the unit.
- Four modes of stepping the motor are supported:
  - Half steps (alternates 1 winding and two windings enabled at a time),
  - Full power full steps (2 windings enabled at a time)
  - Half power full steps (1 winding enabled at a time)
  - Microstep (programmable to as small as $1/16^{th}$ steps, using a near-constant-torque PWM algorithm)
- Motor coordinates are maintained as 32 bit signed values, and thus have a range of -2,147,483,647 through +2,147,483,647.
- The code automatically scales motor motions between the four motors, so that the 'fastest' motor operates at the target rate which you specify, and so that all motors arrive at their target destinations at effectively the same time.
- Permits 'chaining' of goto-vectors, so that you don't have to stop between each line
- A TTL "busy" signal is available, which can be used to see if the motors are still moving. This information is also available from the serial connection.
- Complete control of the motors, including total monitoring of current conditions, is available through the 9600 baud USB serial connection (2400 to 19,200 baud available).
- Runs off of a single user-provided 6.5 to 15 volt DC power supply, or two supplies (5V for the logic circuits and 6.5-26V for the motors).
- 'Slew' inputs may be arbitrarily reprogrammed as generic TTL inputs and outputs

Firmware Configuration

The NCStepper4D firmware has a set of initial settings that are selected at power-on or reset that may be reconfigured at the time the product is ordered. With the exception of the serial baud rate used, all of these features may be reset through use of the appropriate serial command.

Default Serial Baud Rate

All serial communications with the NCStepper4D firmware normally operates at 9600 baud, 8 data bits, 1 stop bit, no parity, no handshake of any form (hardware or software). The communications is normally performed using USB; however, TTL-serial is also provided to allow other techniques of serial communication. As a special factory-order option, 2400, 4800, 9600 or 19200 baud may be requested.

Please refer to the section entitled "Board status and TTL Serial" for information on where to find these signals.

Default Stop Rate

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

Default Microstep Size

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

Default Slew Rate

Normally, the firmware defaults to a slew rate of 800 steps/second (equivalent to the '800r' command). This can be ordered as any valid slew rate for the system.

Default Ramp Rate

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

Default Full-Power Level (FUL jumper installed)

Normally, we ship the product such that the default code will operate at full power (see the 0'H' command) when the board is reset or powered on and the "FUL" jumper (if available on your artwork) is installed. If your artwork only has the 'LOW' jumper available, then you simply do not install it in order to operate at full power levels.

At the time of ordering the product, you may specify the 'H' mode use by default in this case.

Default Low-Power Level (LOW jumper installed)

As with the Full-Power-Level, we also provide a lower power level if the LOW jumper is installed when the board is reset. Normally, this level is set to ½ power; you may order the board such that this jumper is ignored, if that would be preferable.

Note that the 'FUL' and 'LOW' jumpers are mutually exclusive: you install one jumper in either the 'FUL' or 'LOW' position. If your artwork only has the 'LOW' jumper available, then you simply do not install it in order to operate at full power levels.

Default Motor Idle Winding Current

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

Default Verbose Mode

Normally, at power on or reset, the "verbose" mode of the firmware is set to be 'Send CR/LF upon reception of a command, enable fast command response" (equivalent to the '1V' command). At the time of ordering the product from us, you may specify any of the valid settings for the 'V' command (0, 1, 2, or 3).

Escape Character during Serial Routing

Normally, the 'escape' character used to block interpretation of the next character seen during routing (so that binary data can be transmitted to a child board) is set to be '\'. This may be redefined to be any non-command character at the time of ordering the board. For example, the more common 'ESC' character ESCAPE (decimal value 27) may be selected as the one used by the code.

Default I/O Port Directions

Normally, the Slew Input ports are both set to all inputs. The default power-on directions may be set as an order option (the option specifies the contents of the 'F' command, which defines the port directions).

If you are going to be changing the 'Slew' inputs to be outputs, you need to request that the 1K pull-up resistors not be installed on those signal lines. Otherwise, you run the risk of overheating the microprocessor chip, which can cause the lines to stop responding.

Default initial I/O Port Values

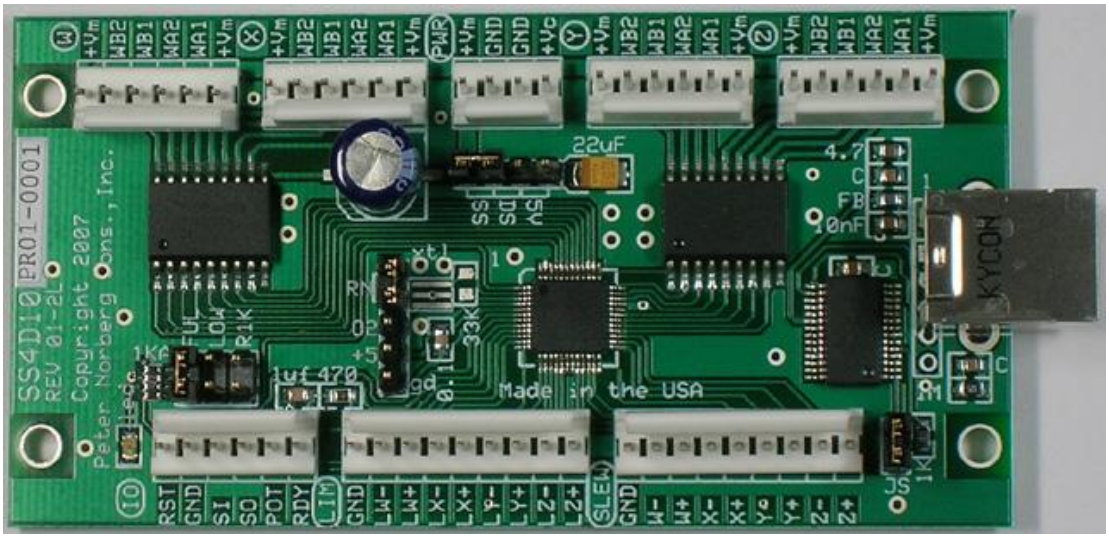
By default, the output-holding registers for the programmable I/O ports are configured for glitch-free operation, with all values set to high (to match the pull-up resistors). These values may be ordered as set to any desired pattern: the option specifies the power-on value for the 'E' command).

Hardware Configuration: Board Jumpers

The NCStepper4D firmware has several major features that can be configured as startup options through use of hardware straps. This means that any combination of these features may be automatically controlled whenever the firmware receives a power-on, hardware reset, or software reset action. In two cases, the features are selected by adding shorting plugs to jumper positions on the board. In the other cases, the features are selected by grounding groups of input pins.

Board Jumpers and Fan Power identification

The following picture shows the SS4D artwork revision 01 board, which shows the most commonly used jumper options.



These jumpers (which may be hard-wired at the factory, or may be present as actual 0.1" shorting-plugs) control various board features. There are 5 jumpers available on the board which control specific operation of the product.

Jumper JS – Enable USB based serial communications

The **JS** jumper is located at the bottom right of the board. If installed, then USB communications via the USB connector (or its optional MTA-100 replacement) connector are enabled. You must NOT use the SI and SO connections when JS is installed, since you will end up with 2 devices driving the same signal (SI), which can eventually destroy one or both devices.

If this jumper is removed, then only TTL-Serial communication will work, via the SI and SO connections.

Jumpers LOW and FUL – Enable Low or FULL Power Mode

The LOW and FUL jumpers are located just above the "IO" connector, near the lower left side of the board. Note that some artwork versions only have the 'LOW' jumper available: on those artworks, full power is selected by not installing a shorting plug on the LOW jumper position.

For artwork versions that have both jumpers available, one shorting plug **must** be installed in one of these two positions: if no jumper is installed, then the board power-on level is undefined. If you somehow jumper both at the same time, you will be shorting power to ground on the board and **you will damage your unit**! This would not be a warranted failure.

Installing the shorting plug onto location 'LOW' will cause the board to power-on (and reset) to operate in the low power mode of operation.

Installing the plug onto location 'FUL' will cause the board to power-on to full power operation. If the 'FUL' jumper position is not present on your artwork, then simply not installing the 'LOW' jumper will cause the board to operate at its full power level.

Note that the actual power levels used for full and low power are controlled by your motor's winding resistance, and the voltage of your motor power supply.

Jumper R1K – UNUSED

The R1K jumper is located just beside the S1K jumper. It is not currently used on the NCStepper4D firmware.

RN Jumper – Must always be installed

The RN jumper is located as the top 2 pins of the 5 pin header in the center of the board. This jumper enables the clock to the microprocessor, and must be installed for the board to operate.

Power input Jumper – SS, DS or 5V

The power input jumper defines how you are providing power to the logic portion of the board. You have three options:

- 'SS' – You are using one supply (via the +Vm/GND input) to power all four of the motors and the board logic. Your +Vm voltage must normally be between 6.5 and 15 volts for this to operate reliably.
- 'DS' – You are using two power supplies, one for the motors (the +Vm input), and one for the logic (the +Vc input). The +Vc supply must be between 6.5 and 15 volts, while the +Vm stays between 5 and 26 volts.
- '5V' –You are using two power supplies, one for the motors (the +Vm input), and one for the logic (the +Vc input). The +Vc supply must be exactly 5 volts, while the +Vm stays between 5 and 26 volts.

In all cases, your power supplies must provide regulated, clean DC voltages of the appropriate levels as required by the jumper positioning.

If you operate in the '5V' mode, please be aware that **you can trivially destroy the board** by supplying a voltage which is not well-regulated 5V DC. **At anything above 6 volts, the board will definitely be destroyed quite quickly**, while anything above 5 volts will cause the SX microprocessor to run hotter (and possibly overheat). If the voltage ever drops to 4.3 volts or less, the microprocessor will act as if it has had a power failure – it will lose all settings and perform a restart action.

Cooling Requirements

Note that if the current requirements are over about 0.4 Amps/winding for the SS4D05 or over 0.8 amps/winding for the SS4D10, or if you intend to leave the windings on when the motor is idle (see the 'N' command), or if the motor voltage supply exceeds 19 volts, then fan-based cooling of the board is usually required. The driver components can get quite hot, and external cooling will increase their lifetime considerably! You should also use fan-based cooling if you are going to be operating the board in a warm environment (>100 degrees F), or if the board is running "hotter" than you like.

Fan based cooling should be done such that the airflow is directed toward the top or bottom surface of the board, centered over the UBICOM chip. Each fan should provide about 6-10 CFM of air flow. Note that the board includes mounting holes positioned such that two 1.6 inch (40 mm) fans may be directly mounted, each through use of two #4 standoffs. If the fans are mounted facing down at the top of the board, use 1 inch standoffs. If mounted facing up at the bottom of the board (which cools better), use ½ inch standoffs. You may wish to review our section "Install The Fan Assembly", located at the end of this manual.

Power-On (and reset) Defaults

In addition to the above hardware straps, the board acts at power on (or reset) as if the following serial commands have been given (note that most of these commands can be overridden through options at the time of ordering product, and through hardware straps):

- **4!** – Microstep size is set to 1/16th of a step.
- **15M** – Select all four motors for all actions (such as defining motor current)
- **0**= – Define all motors to be at location 0
- **255E –** Set all output holding registers to '1', for glitch-free switchover to output mode if needed.
- **0F –** All programmable ports are configured as inputs.
- **0H –** All motors run at full power
- **80K** – Set the "Stop OK" rate to 80 steps/second
- **0N** – All motors are fully off when idle
- **3O** – Enable microstepping mode for the motors
- **8000P** – Set the rate of changing the motor speed to 8000 steps/second/second
- **800R** – Set the target run rate for the motor to 800 steps/second
- **0S** – All motor backlash values set to 0
- **0T** – Enable all limit switch detection
- **1V** – Set <CR><LF> sent at start of new command, no transmission delay time

USB Driver Installation Under Windows

FTDI (http://www.ftdichip.com) provides drivers for operation under Windows[tm], Linux, and Mac/OS. Our installation disk includes snapshot copies of their Windows[tm] drivers; however, you instead may prefer to go to their web site to obtain the most up-to-date revisions of their code. All Linux and Mac/OS customers must download their drivers directly from the http://www.ftdichip.com site, as we have no support capability for those platforms. Look for the drivers and documentation that relate to their FT232RL device.

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

Base Driver Installation Under Windows

Installation of the drivers under Windows is fairly straightforward (and is well documented by FTDI). You should review the FTDI documentation (specifically, their "Windows Driver and Installation Guide" manual) for a more comprehensive description of the installation process. We install a snapshot version of this manual as part of our installation process, but you may wish to obtain a more up-to-date version directly from their web site.

A short summary of the procedure follows, along with a description of the adjustments that should be made to the COM emulator port settings after installation has been completed.

1. Thanks to the "magic" of "Plug-N-Play", connect the SS4D board to your computer (use a normal USB A-B cable of the appropriate length, connecting the 'A' side to your computer USB slot, and the 'B' side to our board). ***Make certain that the SS4D board is NOT on any sort of conductive surface (for example, metal, your hand, a carpet) when you do this, since you could damage the board (or your computer!) if any of the signals on the board get shorted.*** Note that you do NOT need to have the board connected to any external product (such as an actual motor driver) to install the drivers: just the SS4D board and a USB A-B cable are needed, in addition to your computer with its USB 1.1 or 2.0 connection.
2. The SS4D artwork powers the USB portion of the board from the power available from the USB connection itself, and thus does not require board power for Windows to start the driver installation process.
3. This will cause Windows to bring up their "Found New Hardware" wizard, which will guide you through the installation process.
4. Place our installation CD into your CD drive.
5. If our setup application starts up, cancel out of it
6. Tell the wizard to "search for a suitable driver", and then tell it to "specify a location".
7. It will then ask for where to search: tell it to look in the "FtdiStepperBoard" directory on our support CD. Note that we also have an 'ftdi' directory present; FtdiStepperBoard contains modifications to their installation script which simplifies your installation of our products.

8. Then tell it to install the driver. If you are installing from the 'FtdiStepperBoard' version of the drivers, then Windows will complain that the drivers are not 'Windows Certified'. You may ignore the error; all that is different between the 'ftdi' certified installation and the 'FtdiStepperBoard' non-certified installation is that the installation script sets the communication defaults to our recommended values (below).
9. The installation may then go through the same process in order to install the virtual COM drivers (if you have never installed an FTDI USB-based product before). Use the same subdirectory and process to install those drivers as were used under step 7, above.
10. Once that process completes, the code will automatically add a new "COM" serial port that is "attached" to the board when it is plugged into the ***any*** USB port on your computer. *The system will automatically add a new COM port each time you attach a new board to any USB port on your computer or hub. It may also create a new COM port if you receive a repaired board back from us (if we have had to replace the USB driver chip).*

Adjusting Default COM port properties for best operation

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

1. Under Windows 2000 or XP, go to your "system properties" page. Do this by
    a. Right-click on your "System" icon
    b. Select "Properties"
    c. Select "Hardware devices" (it might just be called "Hardware")
    d. Select "Device Manager"
2. Look under "Ports (COM and LPT)", and select the COM port that you just added (it will normally be the highest-numbered port on the system, such as "COM6"), and edit its properties. Note that the 'TestSerialPorts' application will have identified this COM port for you as part of its report.
3. Reset the default communication rate to:
    a. 9600 Baud,
    b. No Parity,
    c. 1 Stop Bit,
    d. 8 Data Bits,
    e. No Handshake
4. Select the "Advanced Properties" page, and set the:
    a. Read and Write buffer sizes to 64 (from their default of 4096).
    b. Latency Timer to 1 millisecond
    c. Minimum Read Timeout to 0
    d. Minimum Write Timeout to 0
    e. Serial Enumerator to checked
    f. Serial Printer to unchecked
    g. Cancel If Power Off to unchecked
    h. Event On Surprise Removal to unchecked
    i. Set RTS On Close to unchecked

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

Initial testing of SS4D board after driver installation – TestSerialPorts

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

The test assumes that the board is correctly configured to 'talk' to the com port: in the case of the SS4D board using its on-board USB driver, the 'JS' jumper must be installed for the TestSerialPorts application to be able to locate the board. Additionally, the board must be correctly powered (USB power alone is not enough; you also must power the 'brains' of the board using your own power supply).

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

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

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

TTL Signals

The TTL input system normally provides for 8 dedicated limit input signals and 8 user programmable input/output signals (which may be individually redefined as outputs, if that is required for your application (see the 'F' command on page 34 for more information)). Additionally, TTL versions of the serial input and output lines (SI and SO) are available, as is a 'RDY' output which tells you whether the motors are idle.

All external connections are done via labeled terminal block connections on the "left" and "right" hand sides of the boards, and one USB serial port on the "bottom" of the board. Most of the input and control signals are on the left side, while all of the motor and power connections are on the right side, as are some generic TTL I/O lines.

TTL Output Current Levels – keep to no more than 5 mA per output

When configured as output drivers, all of the TTL output signals on the board are designed for low-current operation. Although any individual line has a rated drive current (source or sink) of 20 mA, the real limiting factor has to do with how power is distributed throughout the microprocessor that is generating the signals. It can handle at most 50 mA of current draw from its 5 volt supply for each group of 10 I/O signals; thus, you need to keep your current demands down to an average of 5 mA per line.

Additionally, if you actually do drive signals at noticeable current levels, the SX/48 chip will get warm, and may get hot. You may find that you have to cool the board (see our section entitled "Install The Fan Assembly") if the SX/48 seems to be running too hot.

If you exceed these limits, the SX/48 chip is quite likely to 'hang', and suspend all operations in an attempt to protect itself. It is very probable that the chip will be damaged, as a non-warranted failure. This will result in sudden stopping of your motors, and in failure of any application that is using the system. We strongly suggest that you buffer any outputs whose drive current may exceed the 5mA recommended level, in order to avoid such issues.

TTL Input Voltage Levels: Schmitt-Triggered or CMOS

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

Note also that all of the TTL inputs are always internally tied to +5 via a very weak resistor (of the order of 5K- to 40K-Ohms). They are also usually connected to additional 1K pull-ups (unless specifically ordered without the pull-ups for special configurations). This permits you to use switch-closure-to-board-ground as your method of generating a "0" to the board, with the "1" being generated by opening the circuit.

Input Limit Sensors, lines LW- to LZ+

Lines LW- through LZ+ are used by the software to request that the motors begin to stop moving when they reach a hardware-defined positional limit. Enabled by default at power on, the firmware also supports the 'T' command, which may be optionally used to enable or disable any combination of these switches, as well as to configure the level sensitivity (whether it is high to stop or low to stop).

The connections are:

Signal	***Limit Sensed***
LW-	-W
LW+	+W
LX-	-X
LX+	+X
LY-	-Y
LY+	+Y
LZ-	-Z
LZ+	+Z

The connections may be implemented as momentary switch closures to ground; on the connector, a ground pin is available near the LW- pin. They are fully TTL compatible; therefore driving them from some detection circuit (such as an LED sensor) will work. The lines are "pulled up" to +5V with a very weak (10-20K) resistor, internal to the SX-48 microcontroller. By default, we also include an additional 1K pullup, to 'strengthen' the signal: this extra resistor pack may optionally be excluded from the assembly.

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

As the most extreme example possible:

- if for some insane reason the motor is currently running at its maximum rate of 38,422 steps per second,
- and the allowed rate of change of speed is 1 step per second per second,
- and the stop rate was set to 1 step per second,
- then the total time to stop would be 38,422 seconds (a little over 10.6 hours -- groan!), with a distance of ½ v^2, or ½ (38,422)^2, or 738,125,042 steps.
- Note that this same amount of time would have been needed to get up to the 38,422 rate to begin with…

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

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

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

Once a limit switch action has been triggered and the motors 'paused', the firmware is placed in a 'paused' state. Either the 'r' or 'Q' command must be issued to exit this state – 'r' will (attempt to) restart from the pause (which probably will not work, since the limit switch will probably still be blocking the paused action), while 'Q' will abort the action and all pending requests, allowing you to 'back out' of the limit.

General TTL signals collection 1: W- to RDY

Lines W- through Z+ are used in other firmwares to manually control stepping of the motors. On the NCStepper4D firmware, their use is more generic; the W- to Z+ signals are normally configured as inputs, operated via microswitch closures to ground. Through use of the 'F' command, any combination of these signals can be redefined as outputs, thus providing up to 8 TTL-level output signals under computer control on the left side of the board.

If you are going to be changing most of the 'Slew' inputs to be outputs, you may need to request that the 1K pull-up resistors not be installed on those signal lines. Otherwise, you run the risk of overheating the microprocessor chip, which can cause the lines to stop responding.

The POT input signal is unused in the NCStepper4D firmware.

The RDY output signal is used to provide an indication of motor activity: while HIGH, motors are idle (i.e., the board is ready for new commands). While low, motor actions are pending or under way.

Serial Operation

The TTL and USB based serial control of the system allows for full access to all internal features of the system. It normally operates at 9600 baud, no parity, and 1 stop bit. Note that you should wait about ¼ second after power on or reset to send new commands to the controller; the system does some initialization processing which can cause it to miss serial characters during this "wake up" period. If your system does not provide a TTL-Serial or USB-serial signal, the RS232ToTTL product is available for purchase as a serial level converter.

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

1000XY

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

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

Baud Rate is set to 9600 Baud Except For Special Factory Order

By default, the baud rate on the system is fixed at 9600 baud, no parity, and 1 stop bit. 2400. 4800 and 19200 baud rates are also available as a direct factory order option.

Serial Commands

The serial commands for the system are described in the following sections. The code is case-sensitive in many cases; it is best to treat it as entirely case sensitive to allow for future code extensions. ***Please be aware that any time any new input character is received, any pending output (such as the standard "*" response to a prior command, or the more complex output from a report) is cancelled.*** Additionally, for commands which have automatic waits built into them (such as "G" and "A"), the commands can be aborted, if more actions are still pending.

Serial Command Quick Summary

Most of the commands may be preceded with a number, to set the value for the command. If no value is given, then the last value seen as any form of input parameter is used.

General Commands

I/O Port Direction Control, direct set and clear of outputs

Motor Control Configuration

Motor Selection

Motor Motion Configuration

Motor Arc Drawing

Motor Motion Control

Serial command synchronization

Most serial commands immediately send back an '*' to indicate that they have completed their action. Some commands, however, may take an arbitrary amount of time to execute. The core motion code in the firmware maintains a 3 element queue of pending requests. The A, G, R, W, X, Y, Z, = and binary set data commands all are obligated to wait until there is queue space available before they can operate; therefore, they will not send back their particular acknowledgement of the action (the '*' for R, W, X, Y, Z and =, a complete queue status report for A and G) until the request can execute. This can make it a little difficult for application code to keep synchronized with what is going on, therefore the following rules have been added to standardize behaviours:

1. All commands which auto-wait for queue space permit use of the 'nested poll' commands (h, i, I, r, q, ~); however, any of those poll commands will cause the standard response of the auto-wait command to be abandoned (in order to avoid unexpected data transmissions from the controller).
2. The response for all commands which give any status report (A, G, I, i…) is now standardized as 3 letters

   s#*

   's' is the status after the command starts

   'A' – currently executing an 'Arc' command; special mode, most commands will abort continuation of the arc (except for the above 'nested poll' commands).

   'G' – currently executing one or more 'G' commands

   'I' – no command executing, all queue buffers are empty

   'P' – board is paused ('q' or limit switch)

   '#' is the count of available queue elements (0 to 3, for the current firmware)

   '*' ALWAYS completes ANY response
3. The W, X, Y, Z, R, =, and binary set data (second byte) commands all are auto-wait with an 'accepted' response of '*'. If there is no queue space, they will wait until there is queue space before executing. Note that '=' requires the motors to be completely idle in order to execute.
4. If the system becomes paused and there is no queue space, then a full status report ("P0*") will be sent (formatted per item 2, above) (if you have any pending status report or '*'), and the pending status report (or '*') for the currently executing command will be abandoned. You will need to restart ('r') in order to allow the command to complete. You will have to issue an 'i' to get a status report which tells you that it has completed.
5. For ALL commands that wait, the allowed nested commands are (all case sensitive):
   a. 'h' – immediate status report ('s#*')

b. ‘I’ – queue status report ('s#*') request to be sent WHEN MOTORS ARE FULLY IDLE (or paused)

c. 'i' - queue status report ('s#*') request to be sent WHEN THERE IS QUEUE SPACE (or paused/no queue space)

d. ‘q’ – pause the motor (immediately returns ‘*’)

e. ‘r’ – restart from pause (immediately returns ‘*’)

f. '~' - abandon ANY pending status report or queued '*' (used to resynch communications)

6. When ‘A’ has queued its final vectors, the next ‘h’ command will report ‘G’ status instead of ‘A’ status (may report ‘I’ if the motion completes before requesting a status update).

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

Possible combinations:

-n: Value is treated as -n

n: Value is treated as +n

+n: Value is treated as +n

Examples:

-100X – Set next motor X location to be -100

100Y – Set next motor Y location to be +100

A – Draw an Arc of the requested radius

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

- The W, X and Y and Z commands set the CENTER point of the arc. The last two such commands identify which two motors are to be accessed as the logical 'X' and 'Y' axis, from the point of view of calculating the arc. For example, if the last two motors selected (in order) were "W" and "Z", then the W motor would be selected as the logical 'X' axis for the motion, while the Z motor would be selected as the logical 'Y' axis for the motion.
- '**D**' defines the signed "delta angle per line", where the angle units are "degroids", each of which are 1/256 of a full circle (360/256 of a degree, or 1.40625 degrees)
- '**C**' defines the count of line segments drawn at the 'R'un rate; 0 means just draw a line from the current location of the length requested in the direction defined by the current '**B**'egin angle
- '**c**' specifies the slow-down count of line segments. These get generated after the 'C' count is complete, and are automatically preceded by a request for the motors to run at the 'start/stop' speed ('K' command setting)
- '**B**' specifies the beginning angle (again, in units of degroids, where one degroid = 1.40625 degrees)
- '**A**' specifies the radius of the circle, and actually draws the line

This command can take a very long time, since it operates by drawing the requested number of straight lines in a "circular" pattern. Once it has queued the last vector to be generated, if will send a standard status report (see the 'h' command), which will tell you that the primary processing of the arc has completed. This status report (usually "G0*") is not sent back over the serial line until the last line segment has been queued to be drawn. If any new character except 'h', 'I', 'i', 'r', 'q' or '~' is received by the controller while drawing an "arc", then the arc drawing will be stopped at the next vertex. The 'h'command may be used to see if the arc is still going; it immediately echoes back a status report identifying the main action of the controller. The commands 'q' and 'r' may be used to pause or restart the arc. The '~'

character may be used as a "spacer" for communications timing – the 'A' and 'G' commands ignore it, aside from abandoning any pending status report.

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

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

If you execute the 'A'rc command when the 'R'ate command is set to a positive value (such as "1000R"), then each arc segment is drawn as a completely separate line entity. This means that trapezoidal profiling will be on: for each line segment, the system will start at the start rate, ramp at the ramp rate towards the target rate, and slow back down to the start rate by the time the segment is done. If the 'R'ate value is set to a negative value (such as "-1000R"), then the code will treat the entire arc sequence defined by the 'C' parameter as one long line, without a slow-down at the end. The code will start at whatever the current rate is, accelerate to the desired 'R' rate (absolute value), and then stay at that rate for each arc segment defined by the 'C' command. If the 'c' parameter is non-0, it will then set the target rate to the start/stop rate, and then do 'c' arc segments as it slows down towards the start/stop rate.

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

```
*0X      - Set X and
*0Y      - Y center point for the arc and as the arc-based motors
*-2000R  - Set non-trapezoidal mode, so we get a fast circle
*1D      - Set the Arc-Delta to 1 degroid (1 degroid is 1/256 of a full circle; or 360/256 degrees)
*253C    - Tell the system that there will be 253 steps to draw (253 small lines)
*3c      - 3 lines as slow-down/stop, so we won't break gears
*0B      - Begin "degroid angle" is 0
*1000A   - Radius of Arc is 1000, and draw. This draws a "circle" of diameter 2000 steps.
G0*0X    - Reset center back to 0,0 (otherwise, new center would be the last point drawn)
*0Y
*-2000R  - Set non-trapezoidal mode, so we get a fast circle
*253C    - Reset count to 253; it gets destroyed with each draw
*3c      - 3 lines as slow-down/stop, so we won't break gears
*0B      - Reset arc angle; it is left at last point drawn
*2000A   - Draw a 2000 unit radius circle
G0*0X    - Once again, go to location 0,0 as center
*0Y
*4C      - This time, just set 4 lines in "arc"
*0B      - Again, start at 0 "degroids"
*64D     - set the unit delta to be 64; so that 4 will be a complete "circle"
*2000R   - Set trapezoidal mode, so we get a safe square
*3000A   - Draw the 3000 unit radius arc; since done in 4 steps, it is really a square!
```

G0*

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

*250W - Set W and
*300Z - Z center point to the current location, with ‘W’ being logical ‘X’, ‘Z’ being logical ‘Y’
*0C - Tell the system that there will be 0 steps to draw; we only go to the ‘start’ location
*32B - Begin “degroid angle” is 32 (which is 45 degrees)
*945A - Radius of Arc is 945, and draw. This draws a line of length 945 at a 45 degree angle
G2* - note that the arc just uses up one queue element, since only 1 line is being drawn.

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

B – Select Beginning Arc Angle

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

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

C – Define the arc Count of steps

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

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

c – Arc ‘Slow-Down’ count

‘c’ sets the count of arc segments (see the ‘A’rc command) which are tacked-on to the end of the arc sequence which are used to ramp the motor speed down towards the start/stop rate. This command permits you to specify non-trapezoidal rate processing before you start an arc, which allows all of the vectors of the arc to be drawn at full speed (instead of separate start/stop actions for each vector). The ‘c’ command appends an automatic rate request for the current start/stop rate after completion of the ‘C’ arc-line-segments of the ‘A’rc command, and then draws the requested count of line segments as the rate gets reduced towards the desired start/stop rate.

D – Define the arc Delta angle per step

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

E – Send Data to Programmable TTL I/O Ports

This command is used to send data to any ports on the SS4D which have been defined as output ports through use of the 'F' command (next). Any data sent to a port which is currently configured as an input port will be retained as the value to use when (if) that port is changed to being an output port, thus providing for a defined state change sequence if reprogramming of a port is needed.

The ports are set based on the bits in the data associated with this command. The encoding is:

Bit	*Value*	*Signal: 1=output, 0=input*
0	+1	W-
1	+2	W+
2	+4	X-
3	+8	X+
4	+16	Y-
5	+32	Y+
6	+64	Z-
7	+128	Z+

For example,

32F

would set Y+ high, and all of the rest low, and

255F

would set all output lines high

F – Define I/O Port Directions

This command is used to define the I/O directions for all of the programmable I/O ports. Its data bit-encoded identically to that of the 'E' command. Any bit with a value of '1' defines the associated port as being an output port. Any bit with a value of '0' defines that port as being an input port

Bit	*Value*	*Signal: 1=output, 0=input*
0	+1	W-
1	+2	W+
2	+4	X-
3	+8	X+
4	+16	Y-
5	+32	Y+
6	+64	Z-
7	+128	Z+

For example, to define W+ on the SS4D as an output port, issue the command

2F

To define W+ and Y+ as outputs, you sum the 'values' for W+ Y+ (2 and 32 above, respectively), giving you the command:

34F

If you are going to be changing the 'Slew' inputs to be outputs, you may need to request that the 1K pull-up resistors not be installed on those signal lines. Otherwise, you run the risk of overheating the microprocessor chip, which can cause the lines to stop responding.

G or g – Go to currently requested W, X, Y, Z position

These two commands are used to queue the new W, X, Y, Z location (from the W, X, Y and Z commands) as the next location to target. If the current motor motion mode is trapezoidal (the default), both 'g' and 'G' operate identically (they just queue the next trapezoidal motion). If the current motor motion mode is non-trapezoidal (i.e., a 'R'ate command has been issued using a negative rate), then 'G' will queue a motion request targeting the 'R'un rate, while 'g' will queue a motion request targeting the stop-at ('K') rate.

The software will:

- Wait for the prior "pending" w, x, y, z location to actually be accepted to be drawn
- Calculate the direction and distance of travel for both motors, and the correct relative rates for the actions
- Queue the request to be started upon completion of the current motion
- Send back a status byte describing the controller queue state
- Send back a count of elements available in the queue
- Send back the "*" acknowledgement character

For example,

```
*200W
*1000X
*-25687Y
*43156Z
*G
G3*
```

Would:

1. Set the next W value to 200
2. Set the next X value to 1000
3. Set the next Y value to –25687
4. Set the next Z value to 43156
5. Queue a GOTO on motor location 200, 1000, -25687, 43156
6. Note that the firmware would respond with the queue available count (in this case, '2' queue elements are still available), followed by the '*' acknowledgement.

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

H – Operate motors at ½ power

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

The two settings for this are:

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

1H – Run in ½ power mode

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

h – instant queue and motion status report

'h' always requests an instant report on the current queue and motor status. It always returns a 3 character status response (the same one generated by I, I, G, g and A), as:

s#*

where:

s is the status:

'A' – The board is currently waiting on execution of an "arc" (multi-line segment) request

'G' – The board is currently waiting for a queue slot to enque a new goto target, or is idle

'I' – motion is complete, the board is idle

'P' – The board is 'paused' (due to limit switch actuation or the 'q' command); no further motion will occur until 'r' or 'Q' is issued

is the count of elements left in the motion queue. In the current firmware, this number varies from 0 (queue is full, no new queue commands can complete) to 3 (queue is empty).

* reports completion of the status report.

The 'h' command may be given at any time (including when waiting on a blocked command, such as an 'X' when the queue is already full). It will report the current status, and then the command will continue its execution. Note that if you then want to have a new prompt (an '*') generated when the pending command completes enough for a new command, you would have to follow the 'h' command with an 'i' command, in order to get the delayed completion report.

The "h" command gives you a method of safely resynchronizing with the board – it immediately sends back the complete status report, after which you can decide if you need to send an 'i' or 'I' for a delayed report, or if the controller is ready for new commands.

I – Wait for motor 'Idle'

This allows your code to 'wait' for the motors to be fully idle. It also provides you with the reason as to why the motors are idle. (See the 'h' command for a variant which generates the same response with no wait). Sending an 'I' is always safe (even if you have not received an '*' from another command); this allows you to easily 'poll' the board in a multi-port environment. The command also reports the queue count as part of its return, so that you know how many commands may be buffered.

This is normally used to simply wait for either motion to be complete, or to detect that the motors have stopped due to a limit switch action (they are paused).

The report is sent back once the motors are idle (or paused); it is formatted identically to that of the 'h' instant status report (above).

i – Wait for command queue space

This allows your code to 'wait' for queue space to be available for new motion commands ('W', 'X', 'Y', 'Z', 'G' and 'A'). It also reports what main type of action is occurring; this allows your code to confirm whether a new command can be sent. Sending an 'I' or 'i' is always safe (even if you have not received an '*' from another command); this allows you to easily 'poll' the board in a multi-port environment. The command also reports the queue count as part of its return, so that you know how many commands may be buffered.

'i' waits until there is at least one queue element available (or the motors are paused) before it sends back its report, at which point it sends the same report as that generated by the 'h' command.

J – Set Selected I/O port bits to '1'

This command allows you to selectively set a subset of the bits on the slew lines to high (1). Simply form the correct sum from the following table, to tell the code which set of output bits to set to 1 (high).

The ports are set based on the bits in the data associated with this command. The encoding is:

Bit	*Value*	*Signal*
0	+1	W-
1	+2	W+
2	+4	X-
3	+8	X+
4	+16	Y-
5	+32	Y+
6	+64	Z-
7	+128	Z+

For example,

2J

would set W+ high, and leave all of the rest alone, while

255J

would set all output lines high

K –Set the "Start/Stop oK" rate

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

When a motion starts while in the trapezoidal profile mode, this rate is compared to the requested target rate. If the requested target rate is less, then the target rate becomes the initial rate; otherwise, this rate is used. The motor is then ramped up (using the current slope rate, see the 'P' command) to the requested target rate, and continues at that rate until it is time to ramp back down in order to stop at the requested target location.

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

For example,

100k

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

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

32051K

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

When in the trapezoidal mode of rate control (through use of a positive 'R' command), this action automatically waits for all motor motion to complete before executing. It will not send back its '*' response until the motors are completely idle. If a new (non-'I') character is received before this happens, then the 'K' command is forgotten.

When in the 'non-trapezoidal' mode, this action is instant: the new rate will become effectively immediately.

k –Set the current instantaneous motor rate

If the current rate mode (see the 'R' command) is non-trapezoidal (i.e., a negative rate has been requested), then the 'k' command instantly sets the current rate to the requested value, with no intervening rate ramping. This mode can be used to force start conditions when in the non-trapezoidal mode of motion control.

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	W- limit reached during a W- step action
1	+2	W+ limit reached during a W+ step action
2	+4	X- limit reached during a X- step action
3	+8	X+ limit reached during a X+ step action
4	+16	Y- limit reached during a Y- step action
5	+32	Y+ limit reached during a Y+ step action
6	+64	Z- limit reached during a Z- step action
7	+128	Z+ limit reached during a Z+ step action
8	+256	System power-on or reset ("!") has occurred
9	+512	'G' or 'A' interrupted by a new character receipt
10	+1024	Pause is pending (at least, may also be paused)
11	+2048	System is paused

For example, after initial power on,

```
L
```

Would report

```
L,256
*
```

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

```
L,4
*
```

M – Select multiple motors for following '?' commands

This command selects any combination of the four motors to be selected as targets for any following '?' commands. The value is bit-encoded as follows:

Bit	*Value*	*Motor Selected*
0	+1	M1: W
1	+2	M2: X
2	+4	M3: Y
3	+8	M4: Z

For example,

15M0?

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

At power on/reset, all four motors are selected for the selected actions.

N – Set windings power levels for selected motor when motion is complete, disable motion control if manual mode is selected

The windiNgs command controls whether the currently selected motor(s) has its windings left enabled or disabled once any GoTo or Slew action has completed, and it controls power levels to use during normal stepping. It also permits 'manual' setting of the windings being on or off, to allow control of relay (or similar) devices using the motor driver. It is acted on immediately – that is to say, if the current motor(s) is (are) stopped, then the windings are *immediately* engaged or disengaged as requested.

The values to use for control are bit encoded as follows:

Bits	**Use**
0-1	General mode for the windings when motion is complete. 00 (0) – Windings off 01 (1) – Windings on at 'H' setting 10 (2) – Windings on at ½ of 'H' setting 11 (3) – Windings ALWAYS set to 'fixed' value defined by remainder of this parameter
2-3	Defines how to update 'fixed' winding value when bits 0-1 are set to '3' 00 (0) – Use bits 4-7 as the new set of winding values 01 (1) – Set any bits which are set in 4-7 ON ('OR' with prior bits) 10 (2) – Set any bits which are set in 4-7 OFF (turn off in prior bits) 01 (1) – Toggle any bits which are set in 4-7 ('XOR' with prior bits)
4	(+16) – WA1
5	(+32) – WA2
6	(+64) – WB1
7	(+128) – WB2

Most of the time, the above microcoding will not be used by most applications. They will just use the most common settings of:

- **0N – Full power during steps, completely off when stepping completed (default setting)**

- 1N – Full power at all times (both during steps and when idle; matches 'H' setting)
- 2N – Full power during steps, 50% power when idle (relative to 'H' setting)

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

The extended mode of setting the low 2 bits to '3' permits direct control of the outputs from the motor drivers, so that the device may be used to run relays or other similar products. The setting is overrides any motion command, so 'A' and 'G' modes are blocked for this motor while the motor is in extended manual winding control mode.

When using the extended mode, the low 4 bits of the value define the action to do relative to the current winding settings, while the next 4 bits (bits 4-7) define the actual winding data. You therefore have 4 possible 'manual winding set' base values, of:

- 3 – Set windings to new values as shown
- 7 – Set any windings 'on' which are set in the rest of the data
- 11 – Set any windings 'off' which are set in the rest of the data
- 15 – Toggle any windings on or off which are set in the rest of the data

For example, to set WA1 on, and the rest off, you would add '3' ("Set windings") to the winding value of '16' to form a parameter value of 19, giving the command:

19N

If you then wanted to set WB1 and WB2, you could 'merge' it with the prior value by issuing the '7' command ("OR values") plus the winding bits (WB1 is 64, WB2 is 128), giving the command:

199N

If you then wanted to clear just the WB1 driver, you could issue '11' plus '64', or

75N

(at this point, WA1 and WB2 would be on, WA2 and WB1 would be off).

Finally, to toggle WA1 and WA2 (which in this example would result in WA2 and WB2 being on, WA1 and WB1 off), you could do the 'XOR' command (15) plus the winding values for WA1 and WA2, giving:

63N

O – step mOde – How to update the motor windings

The windings of the motors can be updated in one of three ways, depending on this step mode setting. By default, the code uses "micro step" mode set for 16 steps per complete full step, and performs a near-constant-torque calculation for positions between full step locations. The other modes include two full step modes and an alternating mode. For the full step modes, one enables only 1 winding at a time (low power), while the other enables 2 windings at a time (full power). The remaining mode alternates between 1 and 2 windings enabled.

The values which control this feature are:

- 0 : Full Step, Single winding mode (1/2 power full steps)
- 1 : Half step mode (alternate single/double windings on – non constant torque)
- 2 : Full step, double winding mode (full power full steps). NOTE: This mode is often unreliable at high speeds. You will usually attain a higher speed using mode 1 than you will with mode 2.
- **3 : Microstep, as fine as 1/64th step, constant-torque mode – This is the power on/reset default stepping mode.**

For example,

0o

sets the above ½ power full step mode, while

3o

sets the default microstep mode.

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

- 0: Single winding full step mode: Exactly one winding will be on at a time, and will be on at the selected current for the motor. The "real" physical motor position (in full step units) therefore only updates once every 8 microsteps; thus the "full step" location will be the (microstep location)/8, dropping the fractional part.
- 1: Half step mode: Alternates between having one and two windings on at a time, thus causing the torque to vary at the half-step locations. The "real" physical locations will be at half-step values, and hence the motor will "move" once every 3 microsteps. The "full step" location will be the (microstep location)/8, with fractions of 0 to 3/8 mapping into fractional location 0, and 4/8 though 7/8 mapping into fractional location 0.5.
- 2: Double winding full step mode: Both windings are "on" (at the selected motor current) at a time. As with mode 0, the "real" physical motor position will actually only update once every 8 microsteps. The "full step" location will be the (microstep location)/8, with the fractional part forced to 0.5. Please note that double-winding mode is often not reliable at high speeds.

- 3: Microstep mode. The current through both windings are precision-controlled, so that the microposition can be obtained. The physical motor position expressed in full step units is the (microstep location/microstep size (microsteps/step)).

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

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

The slope can be specified to be from 1 through 38,422 steps per second per second. If a value of 0 is specified, the code forces it to have a value of 8000. If a value above 38,422 (or less than 0) is specified, the code will accept it, but will ramp unreliably (i.e., do not do it!).

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

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

250P500R0X2000YG

would cause the following actual ramp behaviors to occur:

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

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

Q – Stop motors immediately, abort queued motions

'Q' causes the motors to be instantly stopped (no ramping is performed), and all pending queued actions to be aborted. **Please note that this action can damage gear systems, since the stop is immediate, with no slow-down behaviors!**

q – Pause motors

'q' causes the motors to be ramped to a complete stop, according to the current ramp rate and stepping rate. "Stopped" is defined as "having a step rate which is <= the stop oK rate" (see the 'K' command for defining the "stop oK rate"). Once the motors are paused, the system waits for a 'r'estart (i.e., the Restart command) or a 'Q' command (abort all) before it permits any motor motion to occur.

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

This defines the run-rate to be used for the faster motor, and it sets or clears the internal "trapezoidal mode" of motion profiling. The rate may be specified to be between 1 and 38,422 steps per second. If a value of 0 is specified, the code sets the rate to the default value, which is normally 800. If a value outside of the limits is specified, then it is accepted, but the code will not operate reliably. As with the slope ("P") rate, do not specify values outside of the 1-38,422 legal domain.

This defines the equivalent number of steps/second which are to be used to run the faster motor under the next GoTo or Arc command.

If a positive rate is specified (such as "1000R"), then the code will operate in its default full trapezoidal motion profiling mode. This means that each motion request operates as follows:

1. The initial rate is set to the start/stop rate as defined by the current 'K' setting.
2. The target rate is the rate specified by this command.
3. The ramp rate (rate of acceleration) to go from the start rate to the target rate is defined by the current 'P' setting.
4. The code will ramp from the start rate to the target rate, operate at the target rate as long as is possible given the target location, and will then decelerate back to the start/stop rate by the time that it reaches the target location.

This mode of operation allows for fully safe operation of an NC machine, since there will be no possibility of missed steps due to not loading a new target location before the prior one completes.

If a negative rate is specified (such as "-1000R"), then the code will operate in its "chained ramp" mode. This means that each motion request is considered to start at the rate at which the prior motion completed, with the target rate being that defined by this command. This permits chaining of complex motions, without the motors constantly stopping in between each motion. ***However, it can cause damage to the gear/motor system, since it will cause an instant stop of the motors if no new location is specified after a current one has completed!*** Its greatest use is when drawing arcs (the 'A' command): by setting a negative rate before starting an arc, and by defining the stop count ('c') for the arc, the entire arc motion can occur rapidly and smoothly at the requested rate.

Note also that while the rate is negative, the 'k' command gets enabled to reset the current rate of the motor, as opposed to specifying the start/stop rate. This allows you to force initial conditions as is appropriate for your application.

For example,

250R

Sets the stepping rate to 250 steps per second.

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

If you set a positive rate while in the negative rate mode, you will probably get motion problems: therefore, always issue an 'I'dle wait before changing rate modes between non-trapezoidal (negative rate) and trapezoidal (positive rate) motions.

r – Restart motion from a pause ('q' or limit-switch) state

The 'r' command is used to restart motion from a pause state caused by the 'q' command or through actuation of a limit switch during motion. It is a 'nested command', in that it will not abort any pending action; thus, you may restart a paused arc with no bad side effects.

S – Specify the motor backlash amount

'S' is used to specify (in microsteps) the motor backlash amount. The currently selected motor(s) (see the 'M' command) all get their backlash set to the specified value. This value may be from 0 to 65535.

Whenever a motor's direction of spin changes, the motor's position gets preadjusted by the backlash amount so that the system will correctly wind-up (tighten) the backlash. This action is fully transparent to external code: it occurs at any time that a new spin direction is requested for each motor.

Note that the code does NOT insert a separate 'windup' vector to take up the backlash; rather, it increases the number of steps by the backlash amount whenever the motor's spin direction changes.

s – Send SPI Data

SendSPIData is supported by all **NCStepper4D** firmware versions. It allows you to send 1 to 24 bits of SPI data through the Z- and Z+ slew ports, with any of the other slew ports being selected as the chip select bit for SPI.

Wiring:

```
Z-:  SCLK to SPI device
Z+:  DATA in to SPI device
<your selected SLEW bit>: SYNC- (or CS-) to SPI device
```

The board automatically redefines Z-, Z+ and your selected CS line as being outputs, and then sends the SPI data as needed. Upon command completion, the three outputs are left high, and left as outputs.

Special value note: If the selected chip select port is 6 or 7, then no CS action is done by the code. In this case, it is up to you to do your own CS via some other technique.

The data value is bit encoded as follows: if the data ranges shown are exceeded, unpredictable program behavior will result!

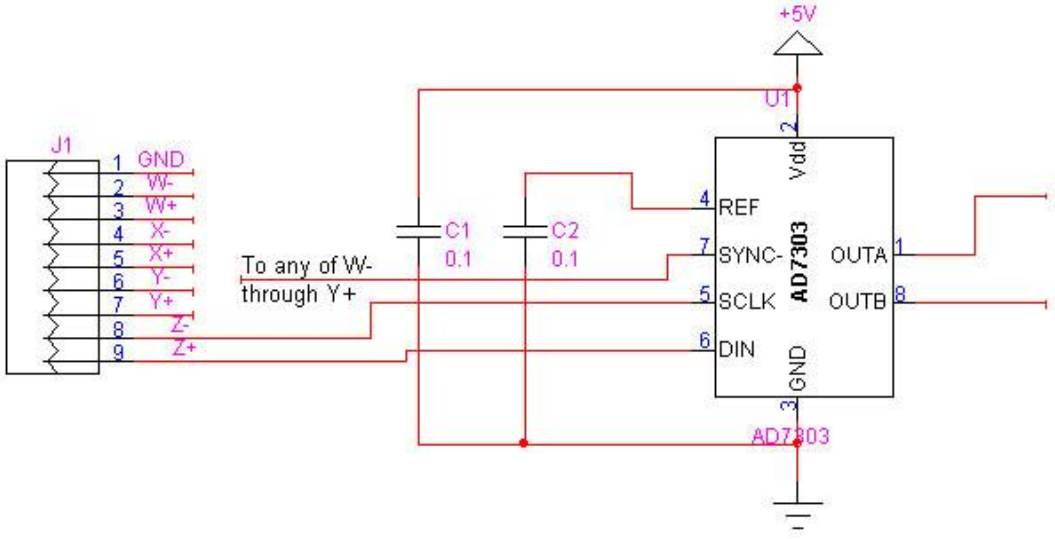
```
Bits    Use
0-4     cDataBits:  1 to 24

5-7     idCSPort:     Value   Use
                        0     W-
                        1     W+
                        2     X-
                        3     X+
                        4     Y-
                        5     Y+
                        6 or 7 No CS auto processing

8-31    lData: 0 to 2^24-1, which is 16777215
```

Example of SPI to an AD7303

The following schematic shows an example using an Analog Devices AD7303 dual DAC as a connection to the SPI system.



In order to set OUTA to a value, you would program it as:

```
cDataBits: 16
idCSPort: 0 (connect SYNC- to W-)
lData bits 0-7: DAC value (0-255 maps into 0 to 5 volts)
lData bits 8-15: microcode for AD7303:
                 3 means load dac A (OUTA)
                 7 means load dac B (OUTB)
```

Thus, to set a voltage of 1 volt on OUTB, we would have the DAC value of 255/5 or 51, and a microcode value of 7. The total encoding becomes:

```
lData: 51 + (7 * 256)
idCSPort: 0
cDataBits: 16

Final value: 16 + (0*32) + (1843 * 256)
                ➔ 471824
```

You would therefore send the command:

471824s

in order to set OUTB to 1 volt.

Similarly, you would send the command:

261904s

in order to set OUTA to 5 volts (lData = 255 + (3 * 256)).

A more generic formula for the 'S' value in this case is:

```
To set either output: Value = BaseValue + DAC * 256
To set OUTA: BaseValue = 16 + (0*32) + (65536 * 3) ➔ 196624
To set OUTB: BaseValue = 16 + (0*32) + (65536 * 7) ➔ 458768

For example: to set a dac to 51 (1 volt), we have:
       OUTA: 196624 + (51 * 256) ➔ 209680S
       OUTB: 458768 + (51 * 256) ➔ 471824S
```

T – limiT switch control

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

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

Bit	Numeric Sum Value	Action
0	+1	Block LW-
1	+2	Block LW+
2	+4	Block LX-
3	+8	Block LX+
4	+16	Block LY-
5	+32	Block LY+
6	+64	Block LZ-
7	+128	Block LZ+
8	+256	Sense level, LW-
9	+512	Sense level, LW+
10	+1024	Sense level, LX-
11	+2048	Sense level, LX+
12	+4096	Sense level, LY-
13	+8192	Sense level, LY+
14	+16384	Sense level, LZ-
15	+32768	Sense level, LZ+
16-31	…	Reserved: leave 0 for now

Note that bits 8-15 are used to define the input level for the indicated limit input lines which are used to stop motor motion. A 0 means "use a logic low to stop", while a 1 means "use a logic high to stop". By default, the system uses a logic low to stop, so that the inputs (which are internally pulled high) will not cause a motor to stop if they are not connected.

For example,

4T

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

65280T

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

U – Set Programmable TTL I/O Bits to 'Low' levels

This command allows you to selectively set a subset of the bits on the slew lines to low (0). Simply form the correct sum from the following table, to tell the code which set of output bits to set to low.

The ports are set based on the bits in the data associated with this command. The encoding is:

Bit	*Value*	*Signal*
0	+1	W-
1	+2	W+
2	+4	X-
3	+8	X+
4	+16	Y-
5	+32	Y+
6	+64	Z-
7	+128	Z+

For example,

2N

would set W+ low, and leave all of the rest alone, while

255N

would set all output lines low

Note that this command does not affect the current I/O direction definitions: defining a bit to be '0' does not change an input bit to be an output bit. To define the port I/O directions, use the 'F' command.

V – Verbose mode command synchronization

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

The firmware actually recognizes and starts to process 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. The firmware handles this via a configurable option in the 'V' command. If enabled, the code will "send" a byte of no-data upon receipt of a new

command character. This really means that the first data bit of a response to a command will not occur until at least 9 bit intervals after completion of transmission of the stop bit of that command (about 900 uSeconds at 9600 baud); for the Basic Stamp[tm] this is quite sufficient for it to switch from send mode to receive mode. The firmware also adds 2 additional "stop" bits to each transmitted character, when this feature is enabled. This is to allow non-buffering microprocessors some additional time to do real-time input processing of the data.

The verbose command is bit-encoded as follows:

Bit	SumValue	Use When Set
0	+1	Send <CR><LF> at start of processing a new command
1	+2	Delay about 1 character time before transmission of first character of any command response. On firmware versions 1.60 and later, add 2 more stop bits to each transmitted character, to allow more processing time in the receiving microprocessor.

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

For example,

0v

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

3v

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

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

W, X, Y, Z – Set "W", "X", "Y" or "Z" value to be used on next "G" command

This command sets the next W, X, Y or Z parameter as will be passed to the next "G" command to the value requested. Depending on the case of the command, the parameter will either be used directly (absolute addressing) or will be added to the current 'W' parameter value and then used (relative addressing).

Upper case commands ("W", "X", "Y", and "Z") set the location as an absolute. Lower case commands ("w", "x", "y" and "z") set the location relative to the prior value.

For example,

100W

Would set the next W parameter value to be 100, while

100w

would set the next W parameter value to the current W parameter value plus 100.

Binary Set W, X, Y, Z and Rate

It is possible to specify the next W, X, Y, Z or Rate value through use of a 2-byte sequence. The delta value may be from –2048 through +2047, allowing for a significant performance savings in terms of the number of characters which it takes to request the next coordinate ('-2048X' takes 6 characters (and hence 6 milliseconds at 9600 baud) to transmit, while the binary version takes 2 characters/2 milliseconds).

The binary set command consists of 2 successive bytes: the first must always have its sign bit set (and is bit encoded to describe which motor is to be accessed, and to contain the top 5 bits of the delta value), while the second byte contains the low 8 bits of the delta value.

The encoding therefore is as follows on the two bytes of binary data:

Byte 1:

Bit(s)	Numeric Value	Description
0-3	0-15	Top 4 bits of the signed 2's complement delta value
4-6	0-7	Encoded ID of binary action to perform. If bit 4 is 0, then the action is a relative W, X, Y or Z value, as: 6 5 4 0 0 0 Assign delta W value 0 1 0 Assign delta X value 1 0 0 Assign delta Y value 1 1 0 Assign delta Z value If bit 4 is 1, then the action is an extended assign 6 5 4 0 0 1 Assign Rate absolute (rate values 1 to 2047) 0 1 1 Assign Rate to value * 16 1 0 1 ** reserved ** 1 1 1 ** reserved **
7	1	Always set this to 1 (+128) to enable this command

Byte 2:

Bit(s)	Numeric Value	Description
0-7	0-255	Low 8 bits of the signed 2's complement delta value

For example, to specify sending a delta X of –1325, the following pseudo-code might be used under VBScript to generate the two bytes of data (this code is not optimized – it merely is intended to show the encoding):

```
Dim chToSend1          ' Character which is being sent first
Dim chToSend2          ' Character which is being sent second
Dim lValueToSend       ' Value to be sent
Dim idMotor            ' Motor (1 for X, 2 for Y)

lValue = -1325         ' Value we are sending
idMotor = 1            ' We are accessing the X motor

' The following section splits up the above request into the two bytes needed

chToSend1 = ((lValueToSend \ 256) And 15) ' Get the masked high 4 bits
chToSend1 = chToSend1 + (idMotor * 32)       ' Include the motor reference
chToSend1 = chToSend1 + 128   ' And include the flag which enables binary data

chToSend2 = lValueToSend And 255      ' Second byte is simply the value masked

' At this point, chToSend1 is the first byte to send,
' and chToSend2 is the second byte to send

' <so send the data…>
```

= – *Define current position for all motors after waiting for motor idle*

This command first waits for all motor motion to complete, and then resets the current location for all motors to be the current scratch values as set by the most recent W, X, Y and Z commands.

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

2000X

4000Y

=

Would define the current location of the X motor to be 2000, and the current location of the Y motor to be 4000. Note that no actual motor motion is involved – the code simply defines the current location to be that found in the associated motor VALUE register.

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

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

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

The suggested values would be the powers of 2, vis. 1, 2, 4, 8, 16, 32 and 64 (giving you true microstep step sizes of 1/64, 1/32, 1/16, 1/8, ¼, ½ and 1 respectively). All other values (such as RATE or GOTO LOCATION) are then expressed in units of the microstep size; therefore, location "3" would mean "3/64" in the finest resolution (microstep set to 1), and "3" in the largest resolution (microstep set to 64).

For example,

4!

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

The reset command also selects the following settings:

- **15M** – Select all four motors for all actions (such as defining motor current)
- **0**= – Define all motors to be at location 0
- **255E –** Set all output holding registers to '1', for glitch-free switchover to output mode if needed.
- **0F –** All programmable ports are configured as inputs.
- **0H –** All motors run at full power
- **80K** – Set the "Stop OK" rate to 80 steps/second
- **0N** – All motors are fully off when idle
- **3O** – Enable microstepping mode for the motors
- **8000P** – Set the rate of changing the motor speed to 8000 steps/second/second
- **800R** – Set the target run rate for the motor to 800 steps/second
- **0S** – All motor backlash values set to 0
- **0T** – Enable all limit switch detection
- **1V** – Set <CR><LF> sent at start of new command, no transmission delay time

? – Report status

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

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

1-255: Report memory location 1-255

Useful locations: ***NOTE THAT ANY LOCATION ABOVE 9 MAY CHANGE BETWEEN CODE VERSIONS***

- 5: Port A register – this contains serial I/O
- 6: Port B register – this contains the limit inputs
- 7: Port C register – this contains the slew inputs
- 8: Port D register – this contains the step-and-direction signals
- 9: Port E register – this contains the IO0 to IO7 signals

0: Report all of items –1 through –11 of the following special reports

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

- -1; Report current location
- -2; Report current speed
- -3; Report current slope
- -4; report target position
- -5; Report target speed
- -6; Report current 'request' scratch-pad value
- -7; report current 'pending' target location
- -8; Report step action (i.e., motor state)
- -9; Report step style
- -10; Report run rate
- -11; Report stop rate
- -12; Report current software version and copyright
- -13 to –17; internal diagnostics
- -18; report the current 'backlash' setting
- -other: reserved – may do extra internal diagnostic reports, or acts like 0 (report all specials)

All of the reports follow a common format, of:

1. If Verbose Mode is on, then a <carriage return><line feed> (“crlf”) pair is sent.
2. The “M” followed by a digit corresponding to the motor being reported on is sent (i.e., ‘M2’ for X, or “M3” for Y).
3. A comma is sent.
4. The report number is sent (such as –4, for target position).
5. Another comma is sent.
6. The requested value is reported.
7. If this is a report for mulitple motors, then a <crlf> is sent.
8. If this is a report for multiple motors, each remaining report is sent.
9. If Verbose Mode is on, then a <crlf> is sent
10. A “*” character is sent.

If multiple motors are being reported, a line for each motor is sent.

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

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

The special reports which are available 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:

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

Where:

- Mx is the motor:
  - M1: W
  - M2: X
  - M3: Y
  - M4: Z
- 0 is the report number; 0 is the 'all' report
- a is the value for the current location (report "-1")
- b is the value for the current speed (report "-2")
- c is the value for the current slope (report "-3")
- d is the value for the target position (report "-4")
- e is the value for the target speed (report "-5")
- f is the value for the current request scratch-pad (report "-6")
- g is the value for the pending target location (report "-7")
- h is the value for the step action (motor state) (report "-8")
- i is the value for the step style (both full step modes and half) (report "-9")
- j is the run rate (report "-10")
- k is the stop rate (report "-11")

For example,

```
6M0?
```

Would report all reportable values for the X and Y motors. You could receive:

```
*
M2,0,30,10,1000,30,10,1000,1000,0,1,100,10
M3,0,-300,10,1000,-300,10,3000,2000,0,1,100,10
*
```

-1: Report current location

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

For example,

```
6M-1?
```

Would report the current location on the X and Y motors. You could receive:

```
*
M2,-1,10
M3,-1,25443
*
```

-2: Report current speed

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

For example,

```
6M-2?
```

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

```
*
M2,-2,800
M3,-2,2502
*
```

-3: Report current slope

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

For example,

```
6M-3?
```

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

```
*
M2,-3,10
M3,-3,25443
*
```

-4: Report target position

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

For example,

```
6M-4?
```

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

```
*
M2,-4,100
M3,-4,-35443
*
```

-5: Report target speed

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

For example,

```
6M-5?
```

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

```
*
M2,-5,800
M3,-5,250
*
```

-6: Report scratch pad value

This reports the current scratch-pad value for this motor. This is the value relative to which all relative motions will be done, and is the value which will be copied as the next 'pending' value when a 'goto' action starts.

For example,

```
6M-6?
```

Would report the current scratch-pad value on the X and Y motors. You could receive:

```
*
M2,-6,1
M3,-6,0
*
```

-7: Report pending target location

This reports the queued target location for the current motor. This will become the target location as soon as the current motion completes.

For example,

```
6M-7?
```

Would report the requested state on the X and Y motors. You could receive:

```
*
M2,-7,64
M3,-7,4
*
```

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

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

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

For example,

```
6M-8?
```

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

```
*
M2,-8,0
M3,-8,4
*
```

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

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

This report currently always returns a '0' on the SS4D series of boards.

-10: Report run rate

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

For example,

```
6M-10?
```

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

```
*
M2,-10,2000
M3,-10,3200
*
```

-11: Report stop rate

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

For example,

```
6M-11?
```

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

```
*
M2,-11,80
M3,-11,50
*
```

-12: Report current software version and copyright

This reports the software version and copyright.

For example,

```
6M-12?
```

could report:

```
*
NCStepper4D.src 1.9
Copyright 2006 by Peter Norberg Consulting, Inc. All Rights
Reserved
*
```

-13 through –17: Reserved

These are reserved for internal use by the firmware.

-18: Report current backlash setting

This reports the current backlash setting for the current motor (the 'H' parameter value).

For example,

```
1M-18?
```

could report:

```
*
M1,-18,231
*
```

~ – *Flush any pending reports*

The '~' character may be used at any time to flush any pending transmissions from the SS4D board. It is used to allow for cleaner resynchronization with the board in the event of use of a SerRoute (serial routing) based multi-board solution, and to guarantee clean message synchronization in the case of retransmitting a queue status request (such as an 'I') when a prior request has not completed.

'~' will not abort any actual action on the controller (except for cancelling pending reports); it is safe to send when waiting on completion of a motion request command, such as 'A' or 'G'.

other – Ignore, except as "complete value here"

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

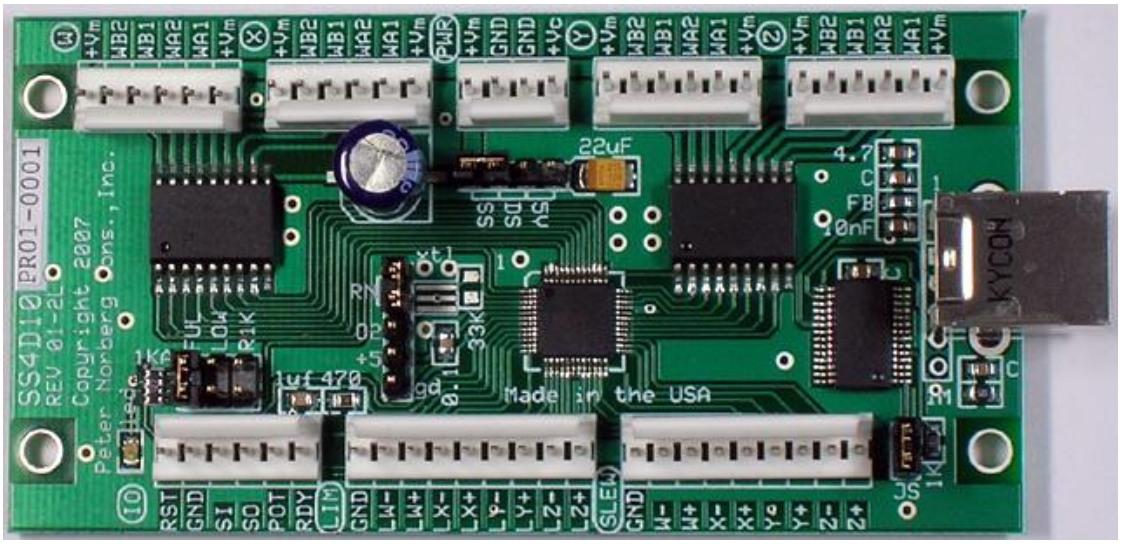
123 456G

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

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

Board Connections

An example of the SS4D board with the MTA-100 connector option is shown below.



SS4D

Board Size

The SS4D board, oriented as shown on this page, is 3.70 inches wide by 1.90 inches high.

Mounting Requirements

The mounting holes are 0.125 inches in diameter, and are positioned exactly 0.125 inches in from the left and right edges, and exactly 0.300 inches in from the top and bottom of the board (as shown in the above image). They allow up to a number 5 screw, which thus allows use of the standard #4 mounting spacers. Horizontally, their centers are 3.45 inches apart, and vertically they are 1.30 inches apart. This separation allows for mounting of cooling fans "directly" on the board (using standard #4 standoffs), without significant extra mounting hardware.

Thus, when the board is positioned as shown above, their positions are:

For the SS4D:
(0.125, 0.300), (3.575, 0.300),
(0.125, 1.600), (3.575, 1.600)

Connector Signal Pinouts

There are ten connectors on each board.

Going from bottom-left to the right, we have:

- SX-Key debugger connector (4 pin SIP header in middle of board)
- SX-48 Direct Access signals:
  - Board status and TTL Serial (RST (Reset), GND, SI (Serial Input), SO (Serial Output), POT, RDY). On the SS4D, only use the TTL serial if the JS jumper, located near the bottom-right portion of the board, is removed.
  - TTL Limit Input (GND, LW- to LZ+)
  - TTL Motor Direction Slew Control (GND, W- to Z+)

Going from top-left to the right, we have:

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

SX-Key debugger connector

Pin	Name	Description
1	GND	Vss (gnd) for SX-Key
2	+5V	+5 for SX-Key
3	OSC2	Oscillator Input 2
4	OSC1	Oscillator Input 1
5	RN	Must be jumpered to OSC1 for the board to operate

This connector allows use of the Parallax, Inc.[tm] SX-Key debugger/programmer product, to reprogram the SX-48 in place. ***If the SX-Key is used as a debugging device, then the 'RN' jumper MUST BE REMOVED, or damage to the SX-Key may occur!***

TTL Limit Input and Reset

Name	Description
GND	Ground reference for inputs – short input to GND to denote limit
LW-	W Minimum limit reached, when low
LW+	W Maximum limit reached, when low
LX-	X Minimum limit reached, when low
LX+	X Maximum limit reached, when low
LY-	Y Minimum limit reached, when low
LY+	Y Maximum limit reached, when low
LZ-	Z Minimum limit reached, when low
LZ+	Z Maximum limit reached, when low

The LIM connector is used to warn the SX-48 that a limit is being reached in the given direction. The signals are designed to be shorted to GND to denote that a limit is present; they are also compatible with normal TTL outputs from any TTL compatible device and are internally pulled up to +5 with 10-33K resistors (within the SX-48 itself).

TTL Motor Direction Slew Control

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

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

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

Board status and TTL Serial

Name	Description
RST	Board reset: ground (and release) this signal to force the board to be reset
GND	Ground reference for all signals
SI	INPUT: Raw SX-48 Serial Input (TTL level)
SO	OUTPUT: Raw SX-48 Serial Output (TTL Level)
POT	Connect to potentiometer to control step rate
RDY	Ready/busy output

This connector gives access to the serial control signals for the SX-48, as well as board status and slew rates.

POT is used to select the rate, through use of a user-provided potentiometer. Please see the "Potentiometer" section at the start of this manual.

RDY is normally an informational output that describes the state of "one or more motors are still stepping". High means READY/IDLE, low means STEPPING. ***During processor reset, RDY is sampled as an input –this can be used on some firmware versions to control certain features.***

SI and SO are the serial input and serial output (respectively), as seen by the SX-48 chip.

The communication rate is fixed at 9600 baud, no parity, 8 data bits, 1 stop bit.

Power Connector And Motor Voltages

Name	***Description***
+Vm	5-26 volts for the motors. If the If the jumper near the power connector is in the 'SS' position, then this is also used as the power input to the on-board voltage regulator to provide logic power to the board. In this case, the voltage must be 6.5 to 15 volts.
GND	Ground for Vwx
GND	Ground for +Vc
+Vc	Supply volts for the logic circuits. If the jumper near the power connector is in the '5V' position, then you must provide 5 volts here. If that jumper is in the "DS" position, then you must provide 6.5 to 15 volts here

There are three ways of powering the logic circuits for system, which can simplify selection of a power supply to use in driving the system.

If you are going to operate your motors with a supply from 6.5 through 15 volts, then one supply may be used to drive all of the motors and the logic supply. To do this, you would set the jumper near the power connector to its "SS" position, and then separately run #22 wires from the +Vm and GND to your power supply.

If you need to "split" the supplies (due to current limitations, or because you need to operate the motors at a value above 15 volts or less than 6.5 volts), then you would create what is called a "star ground". In this arrangement, both of the ground wires from the power connector (the two GND) get separately run with #22 wire to a common heavy-duty tie point, and both of your power supplies get run to that same tie point. Then you run the Vm to its + supply and the Vc to its + supply. For the +Vc, you may elect to either use a supply from 6.5 to 15 volts (in which case you set the jumper near the power connector to the "DS" position), or you may use a 5 volt supply, and set that jumper to its "5V" position.

Be forewarned: if you have the jumper set to the "5V" position, and connect the "+Vc" to a supply which is not a 5 volt supply, you will destroy the board! This abuse is not covered by our warranty!

The power options for the SS4D can be summarized as:

Motor Voltage	***Power Jumper***	***Number of power supplies***	***Comments***
6.5-15V	SS	1	Single power supply, connected to the +Vm and GND power inputs.
5-26	DS	2	Use two power supplies, one for the logic (6.5 to 15 volts connected to the Vc/GND), one for the motors (connected to Vm/GND). Use a "star ground" power arrangement.
5-26	5V	2	Same as above, except that the logic supply is exactly 5 volts.

Note that if the current requirements are over about 0.4 Amps/winding for the SS4D05 or above about 0.8 amps/winding for the SS4d10, or if you intend to leave the windings on when the motor is idle (see the 'N' command), or if the motor voltage supply exceeds 15 volts, then fan-based cooling of the board is required. The driver components can get quite hot, and external cooling will increase their lifetime considerably! You should also use fan-based cooling if you are going to be operating the board in a warm environment (>100 degrees F), or if the board is running "hotter" than you like.

We strongly suggest that you use fan-based cooling if the current requirements exceed the above limits for your board.

Fan based cooling should be done such that the airflow is directed toward the top or bottom surface of the board, centered over the UBICOM/Parallax SX-48 chip. The fan should provide about 6-10 CFM of air flow. Note that the board includes mounting holes positioned such that two 1.6 inch (40 mm) fans may be directly mounted, through use of two #4 standoffs each. If the fans are mounted facing down at the top of the board, use 1 inch standoffs. If mounted facing up at the bottom of the board, use ½ inch standoffs. You may wish to review our section "Install The Fan Assembly", located at the end of this manual.

Calculating Current Requirements

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

1. Determine the individual motor winding current requirements.

This can be determined by reading the specifications for your motor.

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

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

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

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

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

For example, if you are going to microstep (mode 3) a motor whose winding current has been calculated to be 0.4 amps, then your power supply needs to be able to supply 2.5 x 0.4, or 1.0 amps to drive that particular motor.

3. Determine the voltage for your motor power supply

Your motor power supply needs to be above that which would provide the base current through the motor, if the supply were directly connected to the motor. From the formula

V=I*R

where “V” is the power supply voltage, “I” is the current, and “R” is the resistance of the motor windings, we can derive a minimum power supply voltage by knowing the current requirement and by measuring the resistance of the windings. For example, if we have a 0.5 amp/winding motor, which has a coil resistance of 10 ohms, then the minimum voltage for the power supply would have to be 0.5 * 10 or 5 volts. The board requires a minimum of a 6.5 volt power supply, therefore you would us at least a 6.5 volt supply for the motor voltage.

Since the controller is a current regulating controller, using a larger supply voltage than the above minimum (while not exceeding the voltage limits for the board)

simply increases the top speed of the motor, assuming that you have set the 'H' parameter to the correct value for your motor.

4. Determine the logic supply requirements

The SS4D series requires 600 mA for operation if the optional fan assembly is installed. It requires 300 mA if no fan is installed.

5. Determine the power supplies you will be using

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

Single Supply.

If your motor power supply voltage is from 6.5 to 15 volts, then you may choose to use a single supply to operate the system. Position the power jumper at the ">6.5" location, and remember to run separate wires for each of the power connector pins to the supply.

Dual Supply

You may separate the motor supplies from the logic supply. If you do so, you will have the option of providing the 5 volts for the logic system, or of using a 6.5 to 15 volt supply for the logic.

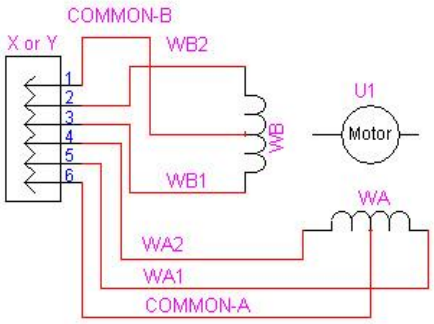
The motor supply should be above 5 volts in all cases (due to some signal requirements on the board), and otherwise is as calculated under sections 1 through 3, above. If the supply is to drive 2 or 4 motors, please remember to double or quadruple the current needs.

ALWAYS REMOVE THE PLACE THE POWER JUMPER IN THE CORRECT POSITION TO MATCH YOUR LOGIC SUPPLY VOLTAGE! The jumper must be in the '5V' position if your +Vc supply is 5 volts, and in the "DS" position if your +Vc supply is from 6.5 to 15 volts.

Always remember that a "star ground" is required when you operate using any split supply; run all three grounds to a common external point, and run the power supply commons to the same point. This avoids referencing issues in the system.

Wiring Your Motor

There are four identical connectors used to operate the W, X, Y and Z motors. The connectors are labeled with respect to which motor they operate. (This designation affects only which commands are to be used to control the motors; no other functionality is changed.) They are wired as follows for the SimStepA04, SS0705, BiStepA05, SS4D and the BiStep2A series of controllers (pins counting from top to bottom):



Typical Unipolar Motor Connection
To the BiStepA05 or SimStepA04

Pin	Name	Description
1	+Vm	+Vm for winding centertap
2	WB2	Winding B, pin 2
3	WB1	Winding B, pin 1
4	WA2	Winding A, pin 2
5	WA1	Winding A, pin 1
6	+Vm	+Vm for winding centertap

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

Stepping sequence, testing your connection

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

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

Determining Lead Winding Wire Pairs

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

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

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

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

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

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

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

You can often operate a 6-wire unipolar motor as if it were a 4-wire bipolar motor (when using the BC2D15 or SS4D series of controllers) by insulating the common leads and leaving them disconnected. When it works, this usually provides more torque for the motor, but it requires double the voltage (at the same level of current) from the power supply. You cannot operate with this pair of wires disconnected if they are connected together inside the stepper motor -- if the resistance between the common leads is not infinite (less than 100,000 ohms), such a connection exists and you must therefore operate using the regular unipolar wiring scheme utilizing a unipolar motor controller.

Sequence Testing

Always double check all of your power and motor connections before you apply power to the system. If you have reversed any power leads, you will blow out our board and you may blow out your power supply! If you are operating a unipolar motor and you short a common lead to a winding pin (WA or WB), then you will blow out our drivers! Similarly, any winding which is shorted to any other winding may burn out our board. None of these issues are warranted failures; repairs for such are not covered!

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

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

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

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

- Issue the command "800R1000XGI0XGI", which should cause the motor to spin to logical location 1000, then back to 0. **Pay attention to the case of the command**: 'r' has a different meaning to the controller than does 'R'.
- Wait for the "*" response after each sub-command (the "R", "X", "G", and "I" commands) before typing the next command, in order to let the firmware finish processing the request.

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

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

A B C D, rewire as

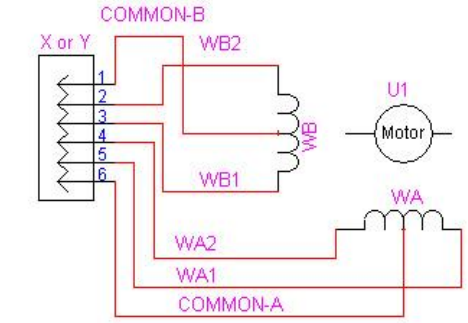
B A D C.

Motor Wiring Examples

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

Unipolar Motors

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



Typical Unipolar Motor Connection
To the BiStepA05 or SimStepA04

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

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

Color	Motor Connector
Black	+Vm
Brown	WB2
Red	WB1
Green	WA2
White	WA1
<no connection>	+Vm

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

This Airpax motor has a manufacturing part number of C42M048A04. As with the other Airpax motor, it does not microstep at all. Mode "3o" can smooth its actions, but it does not "stop" at any other points than ½ step locations. It may be operated using the 1 amp driver SS4D10, but not the 0.5 amp SS4D05! It is wired as:

Color	Motor Connector
Green	+Vm
Black	WB2
Brown	WB1
Yellow	WA2
Orange	WA1
Red	+Vm

When using a 5 volt motor (such as this), you may use a single, 6.5 volt power supply (this may slightly over-voltage the motor), or you may use a split supply. In this case, use a 6.5-12 volt supply for the power to the digital electronics (pins 1 and 4 on the power connector), and a 5 volt power supply for the motor (pins 2 and 3 on the power connector).

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

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

Color	Motor Connector
White	+Vm
Brown	WB2
Yellow	WB1
Red	WA2
Blue	WA1
Black	+Vm

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

This motor provides for 6000 g-cm(!) of holding torque, and has a manufacturing number of GBM 57BYGO84. It may be operated using the SS4D10, but not the SS4D05! Its wiring order is:

Color	Motor Connector
Black	+Vm
Orange	WB2
Green	WB1
Yellow	WA2
Blue	WA1
White	+Vm

Jameco 169201 24 Volt, 0.3 Amp/winding, 1.8 deg/step

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

Color	Motor Connector
Black (Common lead for PEACH and VIOLET)	+Vm
Peach	WB2
Violet	WB1
Yellow	WA2
Red	WA1
White (Common lead for Yellow and White)	+Vm

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

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

Color	Motor Connector
Red	+Vm
Orange	WB2
Pink	WB1
Yellow	WA2
Blue	WA1
<no connection>	+Vm

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

This motor has a manufacturing part number of NMB PM55L-048-NBC7. It may be operated using the SS4D10, but not the SS4D05! Its wiring is:

Color	Motor Connector
Black (common for Brown and Red)	+Vm
Brown	WB2
Red	WB1
Green	WA2
Yellow	WA1
Orange (common for Yellow and Green)	+Vm

Haydon 43F6R-12-005 12 Volt, 0.3 Amp/winding, 1.8 deg/step

This motor has a manufacturing part number of 43F6R-12-005. It uses 6 wires, with the wiring being:

Color	Motor Connector
Black	+Vm
Green	WB2
Green/White	WB1
Red	WA2
Red/White	WA1
White	+Vm

Haydon 43F6R-05-007 5 Volt, 0.7 Amp/winding, 1.8 deg/step

This motor has a manufacturing part number of 43F6R-12-005. It may be operated using the SS4D10, but not the SS4D05! It uses 6 wires, with the wiring being:

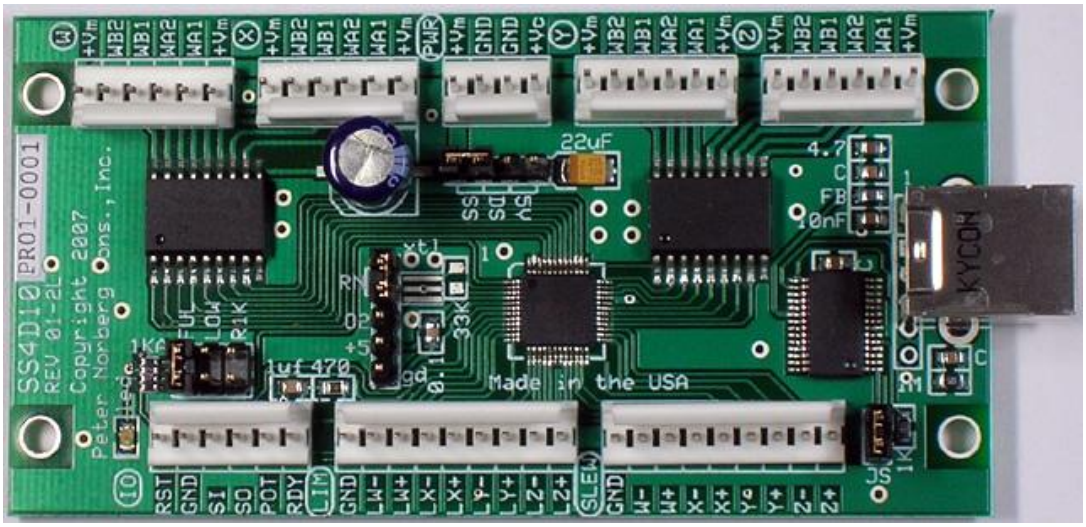
Color	Motor Connector
Black	+Vm
Green	WB2
Green/White	WB1
Red	WA2
Red/White	WA1
White	+Vm

Install The Fan Assembly

When the SS4D unit is purchased, a fan assembly may optionally be included in order to provide cooling for the unit. The fan assembly is shipped separately; it is not mounted onto the board during shipment.

The fan is mounted onto the SS4D via four 1-inch stand-offs, and is powered off of a 5-pin SIP header that is available on the board.

Please review the following image which shows that 5 pin connector near the bottom center of the board.



1. Attach the four 1-inch stand-offs to the SS4D. Use four of the lockwashers and the four nuts to attach the stand-offs; they are positioned as shown in the picture on the following page labeled "Unit with Mounting Posts".
2. Connect the fan power to the power portion of the 5-pin SIP header at the center of the board (this will almost be under the fans once they are installed). The black lead goes to the bottom-most pin (gd), the red lead goes to the next pin on the header (+5). Do not remove the 'RN' jumper from that header! If it is removed or repositioned, the board will not run.
3. Attach the fans to the stand-offs using the remaining screws and lockwashers. **The label side of the fans must face down** as shown in the photo on the next page, labeled "Unit with Fans". ***It is absolutely critical that the direction of air flow be towards the board; otherwise, the board will fail! Any failure induced by incorrect mounting of the fan assembly is not covered by the warranty.***

Figure 1: Unit with mounting posts

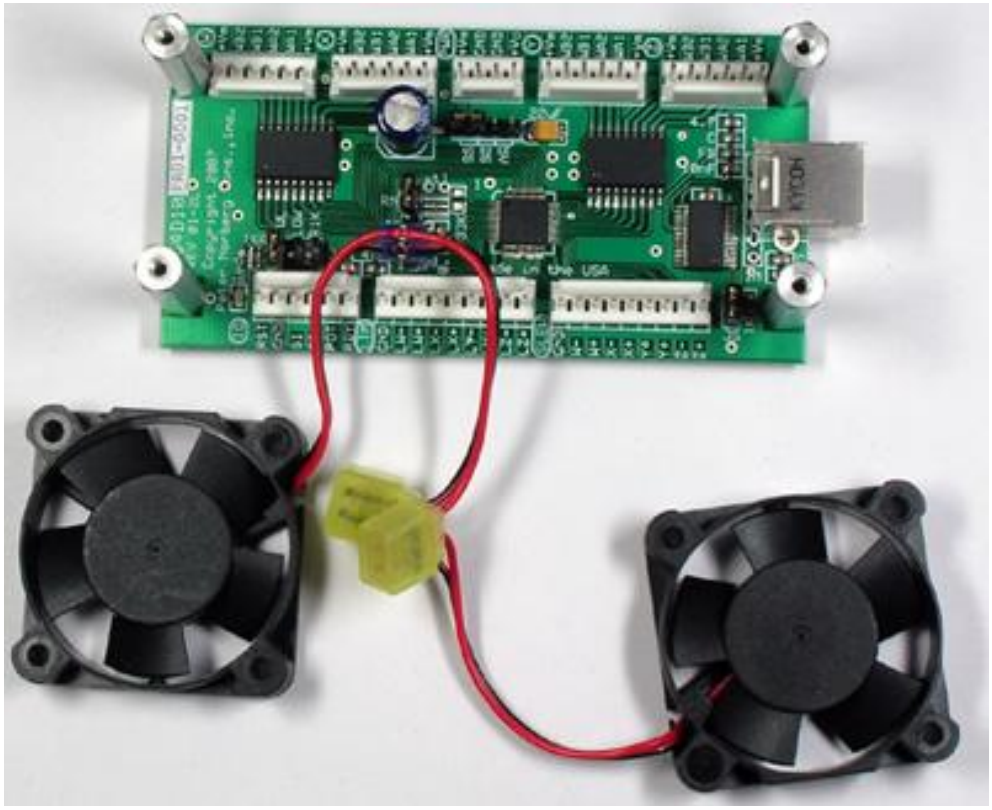


Figure 2: Unit with fans attached to fan power plug

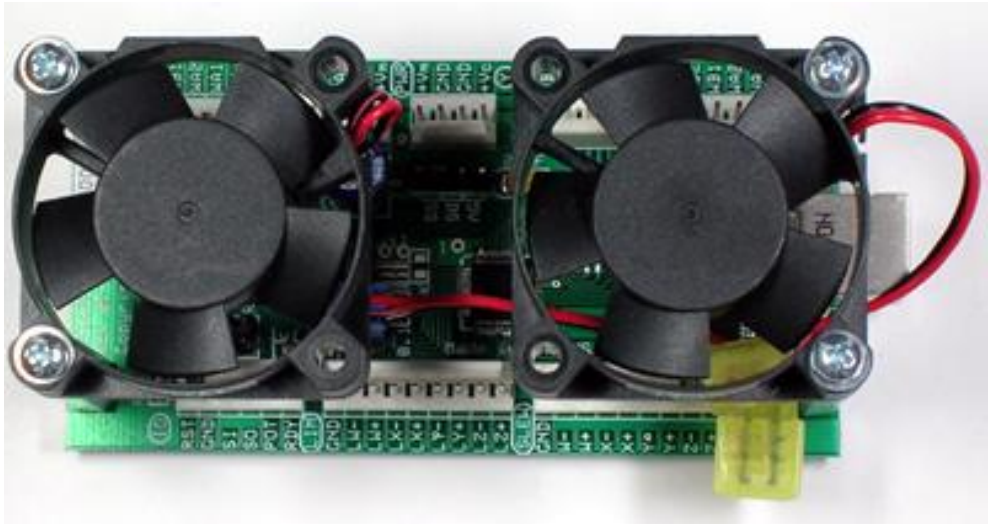


Figure 3: Unit with fans fully installed

Please note that the direction of air flow must be towards the board. If the fans are mounted such that the air flow is away from the board, then the board will fail!

You should (hopefully) now have an operational board. Apply power, and observe whether the LED illuminates, ***and the fans blow air towards the surface of the board***. If either action does not occur, turn the system off, and trace the problem.