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

BC2DNCStepper Firmware for the BC2D15 Stepper Motor Controller

By
Peter Norberg Consulting, Inc.
Matches BC2DNCStepper Firmware Revision 1.9

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 the BC2DNCStepper firmware, release version 1.9.

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

Version	Date	Description
1.6	May 30, 2005	Initial Release
	August 10, 2006	Added notes relating to USB version of the BC2D15 board.
	March 20, 2007	Added minor Vista notes
1.9	October 31, 2008	Improved serial system recovery on bad serial input data

The BC2DNCStepper firmware is designed to operate correctly with the BC2D15 stepper motor controller.

Product Safety Warnings

The BC2D15 series of boards have components that can get hot enough to burn skin if touched, depending on the voltages and currents used in a given application. Care must always be taken when handling the product to avoid touching these components:

- The small LM2940 5 volt regulator (located on the 'bottom' of the board, directly beside the DB9 serial connector)
- The large IMT902 IC on the bottom of the board.

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

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

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

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

If you are operating motors that require more than 0.7 Amps of current per winding or if the motor power supply voltage exceeds 19 volts, then you must use our fan-based cooling of the board. The airflow from the fan must be downward towards the board.

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 **BC2D15** microstepping motor controller from Peter Norberg Consulting, Inc., has the following general performance specifications:

	BC2D15	**BC2D15USB**
Unipolar Motor	No	No
Bipolar Motor	Yes	Yes
Maximum Motor supply voltage (Vm)	34V	34V
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)	1.5A	1.5A
Board size	2.1" x 2.5"	2.1" x 2.5"
Dual power supply capable	Yes	Yes
RS232 communications	Yes	No
USB communications	No	Yes
Available in ROHS compliant version	No	Yes

Each board is to be controlled via its 9600 baud serial interface. Its stepping rates range from 1 to 44801 microsteps per second, its slope rates range from 1 to 44801 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 BC2DNCStepper firmware shares many of the features of the BC2DPotStepper dual-motor controller firmware. The PWM control of the motors 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 BC2DNCStepper firmware explicitly controls both 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 BC2DPotStepper explicitly controls the two motors independently, so that one motor may be performing a "slew" operation, while the other 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
*1d - Set the Arc-Delta to 1 degroid (1 degroid is 1/256 of a full circle; or 360/256 degrees)
*256c - Tell the system that there will be 256 steps to draw (256 small lines)
*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
*256c - Reset count to 256; it gets destroyed with each draw
*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"
*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. 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

- Two Bipolar stepper motors are to be controlled at one time.
- A firmware factory option allows each motor to be operated at a different base current per winding.
- Limit switches may be used to automatically request motion stop of motion if either motor reaches a limit in either direction.
- Rates of 1 to 44801 microsteps per second are supported.
- Step rates are changed by linearly ramping the rates. The rate of change is independently programmed for each motor, and can be from 1 to 44801 microsteps per second per second.
- All motor coordinates and rates are always expressed in programmable microunits of up to $1/16^{th}$ step. Changing stepping modes between half, full and micro-steps does not change any other value other than which winding pairs may be driven at the same time, and how the PWM internal software is operated.
- Motor coordinates are maintained as 32 bit signed values, and thus have a range of -2,147,483,647 through +2,147,483,647.
- Both GoTo and Arc (actually, multi-line-segment) actions are fully supported.
- Arc vertex locations are calculated to a precision of about 1:10,000,000
- 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 2400 or 9600 baud serial connection.
- Any number of motors may be run off of one serial line, when used in conjunction with one or more SerRoute controllers.
- The standard firmware allows generic output control for the IO6 and IO7 lines, providing for two TTL signal outputs under user control. This option may be replaced with the option of specifying the motor currents for the X and Y motors separately (instead of using one value for both motors).

Firmware Configuration

The BC2DNCStepper firmware has a set of initial settings that are selected at power-on or reset that may be reconfigured at the time the product is ordered. With the exception of the mode of stepping used when the "Auto-full-step" rate is reached, all of these features may be reset through use of the appropriate serial command.

Default Microstep Size

Normally, the firmware defaults to a microstep size of $1/16^{th}$ of a full step (the equivalent of the "1!" 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 (1/16, 1/8, ¼, ½ or full-step).

Default Stop Rate

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

Default Ramp Rate

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

Default Full-Power Level (S1K jumper removed)

Normally, we ship the product such that the default code will select a winding current of full rated power (see the '220H' command) when the board is reset or powered on and the S1K resistor is removed. At the time of ordering the product, you may specify the current to use by default in this case.

Default Low-Power Level (S1K jumper installed)

As with the Full-Power-Level, we also provide a lower power level if there is a jumper installed at the S1K position when the board is reset (equivalent to the '55H' command). Normally, this level is set to ¼ of the rated board power; you may order the board with this setting at any current level that is appropriate as an alternate use value for your application.

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

Default Motor Idle Winding Current

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

Default Limit-Switch Stop Mode

Normally, the firmware defaults to treating a limit-switch input as 'soft'; that is to say, the firmware issues a 'z' command when a limit is reached. This can be ordered as a 'hard'

stop – the board will **INSTANTLY** stop the motor when a limit is reached. Note that damage to gear trains is possible if this option is ordered!

Default Limit-Switch Level Sensitivity

Normally, the firmware defaults to treating a limit-switch LOW value as 'STOP'. At the time of ordering, this can be reconfigured to be a 'HIGH'.

Hardware Configuration

The BC2DNCStepper firmware has two features that can be configured as startup options. This means that any combination of these features may be automatically controlled whenever the firmware receives a power-on, hardware reset, or software reset action. Both features are selected by adding jumpers on the board.

Configuring Serial Baud Rate

By default, all serial communications with the BC2DNCStepper firmware operate at 9600 baud, 8 data bits, 1 stop bit, no parity. If you need to communicate at 2400 baud, you may install a jumper in the 'R1K' position on the board.

This feature is available in all versions of the BC2DNCStepper firmware.

Power-on at ¼ Power (equivalent to the "55H" command)

Powering on at half-power allows you to operate motors that are rated at ¼ of the board specifications without having to issue the "55h" command. For example, if you have a 0.3 amp motor, this mode allows the board to automatically power on at the 0.3 amp level.

This mode may be configured by installing the 'S1K' jumper. The hardware selection may be changed at any time through issuing the appropriate "h" commands, as described elsewhere in this manual.

Board Jumpers

The BC2D15 and BC2D15USB boards have a series of jumpers (which may be hard-wired at the factory, or may be present as actual 0.1" shorting-plugs) which control various board features. As shown , there are 4 jumpers available on the board which control specific operation of the product.

Jumper JS – Enable USB or RS232 based serial communications

The **JS** jumper is located at the bottom of the board, near the DB9 RS232 or the USB-B connector. If installed, then RS232/USB communications via the associated 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.

Jumper S1K – Enable Low-Power Mode

The S1K jumper is located near the 'IO' connector, near the lower left side of the board. If it is installed, then the board will power-on (and reset) to operate in the low power mode of operation. If it is not installed, then the board will power-on to full power operation. Note that the actual power levels used for full and low power may be specified at the time of ordering your board. By default, full power is 1.3 amps/winding, and low power is about 0.35 amps/winding.

Jumper R1K – Select 2400 or 9600 baud communications

The R1K jumper is located just beside the S1K jumper. If it is not installed, then serial communications will default to 9600 baud. If it is installed, then communications will normally operate at 2400 baud. Note that these rates may be different if a special order option has been requested.

Jumper SS, DS, 5V – Configure Power Supply mode

This set of jumper options is used to configure the board to accept the power supply voltages that you will be providing. **If you do not provide voltages that match the expectations of the jumper setting, then you may damage the board (and void your warranty)!**

Note that some boards just have the 'SS' position available, while later boards have all three 'SS', 'DS' and '5V' options available.

Please refer to the manual section entitled "" for information on how to set this jumper.

Cooling Requirements

Note that if the current requirements are over about 0.7 Amps/winding, or if you intend to leave the windings on when the motor is idle (see the 'W' command), or if the motor voltage supply exceeds 19 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 0.5 amps per winding.

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. The fan should provide about 6-10 CFM of air flow. Note that the board includes mounting holes positioned such that a 1.6 inch (40 mm) fan may be directly mounted, through use of two #4 standoffs. If the fan is 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:

- **2=** – Select the next "G"oto assigns the location instead of moving the motors
- **0X** – Set next X will be 0
- **0Y** – Set next Y will be 0
- **220H** – Set motors to 1.3 amps/winding (or to 0.3 amps/winding ("55H") if the hardware strap is enabled)
- **G** – Reset both motors to be at location 0
- **0H** – Set motors to full power mode
- **80K** – Set the "Stop OK" rate to **80** microsteps/second
- **3O** – Set the motor windings Order to "microstep"
- **8000P** – Set the rate of changing the motor speed to 8000 microsteps/second/second
- **800R** – Set the target run rate for the faster motor to 800 microsteps/second
- **1V** – Set <CR><LF> sent at start of new command, no transmission delay time
- **0W** – windings are off when the motor is idle

TTL Input Voltage Levels: 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 <= 2.5 volts, and a logic "1" is generated when the input voltage is above 2.5 volts. 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 internally tied to +5 via a very weak resistor (of the order of 5K- to 40K-Ohms). This permits you to use switch-closure-to-board-ground as your method of generating a "0" to the board, with the "1" being generated by opening the circuit.

Input Limit Sensors, lines LY- to LX+

Lines LY- through LX+ are used by the software to request that the motors stop moving when they reach a hardware-defined positional limit.

The connections are:

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

The connections may be implemented as momentary switch closures to ground; on the connector, a ground pin is available near the LY- pin. They are fully TTL compatible; therefore driving them from some detection circuit (such as an LED sensor) will work. The lines are "pulled up" to +5V with a very weak (10-20K) resistor, internal to the SX-28 microcontroller.

Voltages which are <=2 volts are considered to be "0", while voltages >=3 volts are "1". Voltages in the range of 2.1 through 2.9 are transitional, and will be randomly treated as "0" or "1" by the processor.

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

As the most extreme example possible:

- if for some insane reason the motor is currently running at its maximum rate of 44801 microsteps per second,
- and the allowed rate of change of speed is 1 microstep per second per second,
- and the stop rate was set to 1 microstep per second,
- then the total time to stop would be 44801 seconds (a little over 12.4 hours -- groan!), with a distance of ½ v^2, or ½ (44801)^2, or 1,003,564,800 microsteps.
- Note that this same amount of time would have been needed to get up to the 44801 rate to begin with…

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

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

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

Unused control signals: Y-,Y+,X-,X+,NX – Available as TTL Input signals

The input signalsY-, Y+, X-, X+ and NXT are not currently used by the BC2DNCStepper firmware. Note that they may be read using the "6?" report command, however.

To read the current values for these signals, you may send a request of

6?

The code will respond with:

R,6,nnn
*

Where 'nnn' is the value for the reading. It is bit-encoded as:

Bit	Value	Use
0	1	Y-
1	2	Y+
2	4	X-
3	8	X+
4	16	NXT
(5)	(32)	(RDY)
(6)	(64)	(SI)
(7)	(128)	(SO)

Only the low 5 bits are available as TTL input signals; the remaining are listed for completeness.

RDY Output Signal

The RDY output signal may be used to indicate that motor motion is still being requested on at least one of the motors. When HIGH, then all motion is stopped. When LOW, at least one motor is still moving. This signal is LOW as long as there are pending actions in the line queue.

USB Driver Installation Under Windows for the BC2D15USB board

Our boards use a USB driver chip for communications with your hosing computer. FTDI (http://www.ftdichip.com) provides drivers for operation of these chips under Windows[tm], Linux, and Mac/OS. Our installation disk includes **modified** copies of their Windows[tm] drivers; our newest boards use a unique ID code which prevents them from being recognized by the default FTDI drivers. All Linux and Mac/OS customers must download their drivers directly from the http://www.ftdichip.com site, as we have no support capability for those platforms. They will also have to special-order the boards from us such that we configure them to respond to the standard FTDI drivers. Look for the drivers and documentation that relate to their FT232RL device.

A short summary of the installation of the drivers under Windows 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. If you are installing under Windows Vista[tm], you should read our more complete installation instructions as found in our "FirstUse" document. The key additions to the following list when installing under Vista are that you must be logged in as an administrator, and Vista will give you several extra verification prompts in order to confirm that you really want to do this (say 'Yes').

A short summary of the procedure under XP 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 BC2D15USB 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 BC2D15USB 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 power to install the drivers: just the ***BC2D15USB*** board and a USB A-B cable are needed, in addition to your computer with its USB 1.1 or 2.0 connection.
2. This will cause Windows to bring up their "Found New Hardware" wizard, which will guide you through the installation process.
3. Place our installation CD into your CD drive.
4. If our setup application starts up, cancel out of it
5. Tell the wizard to "search for a suitable driver", and then tell it to "specify a location".
6. It will then ask for where to search: tell it to look in the "FtdiStepperBoard" directory on our support CD.
7. Then tell it to install the driver. 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) and tells the FTDI drivers how to access our products.
8. 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.
9. Once that process completes, the code will automatically add a new "COM" serial port which 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).

Initial testing of any BC2D15 board after driver installation – TestSerialPorts

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

The test assumes that the board is correctly configured to 'talk' to the com port. The '**JS**' jumper must be installed and the board must be correctly powered for the TestSerialPorts application to be able to locate the board.

When TestSerialPorts starts, simply press the "Scan Serial Ports" button (you may safely ignore the other buttons). The application will then perform its scan, and will identify every COM port on your system. It will also 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 (and the FTDI USB drivers must be correctly installed, if the USB version of the product is being used) 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.

Adjusting Default COM port properties for best operation

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

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

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

Serial Operation

The RS-232 based serial control of the system allows for full access to all internal features of the system. It operates at 2400 or 9600 baud, no parity, and 1 stop bit. Note that you should wait about ¼ second after power on or reset to send new commands to the controller; the system does some initialization processing which can cause it to miss serial characters during this "wake up" period.

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. 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 7-8 bit intervals after completion of transmission of the stop bit of that command (about 750 uSeconds at 9600 baud); for the Basic Stamp[tm] this is quite sufficient for it to switch from send mode to receive mode.

Selecting Baud Rate

By default, the baud rate on the system is fixed at 9600 baud, no parity, and 1 stop bit. Operation at 2400 baud may be selected during the initialization process via adding a jumper at location R1K. During reset (i.e., power on, hard reset, or processing of the "!" command), this signal is tested for the presence of the jumper. If it is present then the baud rate is set to 2400 baud. If it absent then the baud rate is set to 9600 baud.

Serial Commands

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

0-9, +, - – Generate a new VALUE as the parameter for all FOLLOWING commands
A – Draw an Arc of the requested radius
B – Select Beginning arc angle
C – Define the arc Count of steps
D – Define the arc Delta angle per step
E – Set the extra IO bits (IO6 and IO7) to desired values
G – Go to the currently requested X, Y position;
OR reset motor X, Y location to be the current X, Y parameter values
H – Set power level to the motors when stepping
I – Wait for motor 'Idle'
K –Set the "Stop oK" rate
L – Latch Report: Report current latches, reset latches to 0
O – step mOde – How to update the motor windings
P – sloPe (number of steps/second that rate may change)
R – Set run Rate target speed for the 'faster' motor
V – Verbose mode command synchronization
W – Set windings power levels when motor is not stepping
X – Set next X value as defined by the current "=" mode
Y – Set next Y value as defined by the current "=" mode
Z – Stop motors
! – RESET – all values cleared, all motors set to "free", redefine microstep. Duplicates Power-On Conditions!
= – Define interpretation of "X", "Y", and "G" commands
? – Report status
Any character above 'z' – stop sending pending output data, then skip the character
other – stop sending output data, then echo the '*' command to complete response

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:

1000x – Set the next X value to 1000

-1000y – Set the next Y value to -1000

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 X and Y commands set the CENTER point of the arc
- '**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; 0 means just draw a line from the current location of the length requested in the direction defined by the current '**B**'egin angle
- '**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. The "*" (ready) character is not sent back over the serial line until the last line segment has been queued to be drawn. If any new character except 'I' or '~' is received by the controller while drawing an "arc", then the arc drawing will be stopped at the next vertex. The 'I' character (see Idle Wait) may be used to see if the arc is still going; it immediately echoes back a status letter identifying the main action of the controller. The '~' character may be used as a "spacer" for communications timing – the 'A' and 'G' commands ignore it.

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 will 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.

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
*1d - Set the Arc-Delta to 1 degroid (1 degroid is 1/256 of a full circle; or 360/256 degrees)
*256c - Tell the system that there will be 256 steps to draw (256 small lines)
*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
*256c - Reset count to 256; it gets destroyed with each draw
*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"
*3000a - Draw the 3000 unit radius arc; since done in 4 steps, it is really a

square!
*

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,

*250x - Set X and
*300y - Y center point to the current location
*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
*

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.

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 – Set the 'E'xtra I/O bits (IO6 and IO7)

This command is available on the default configuration of the firmware. It allows setting of the values for the IO6 and IO7 TTL output signals on the board.

Only bits 6 and 7 are actually sent to IO6 and IO7 (respectively); therefore the acceptable settings are:

0E	Set IO6 and IO7 both low
64E	Set IO6 high, IO7 low
128E	Set IO6 low, IO7 high
192E	Set IO6 high, IO7 high

Note that any other bits in the data are ignored; however, it is better practice to just use the above values, to remain compatible with future firmware releases.

G – Go to currently requested X, Y position; OR reset motor X,Y location to be the current X, Y parameter values.

This is normally used to queue the new X, Y location (from the X and Y commands) as the next location to target. Assuming that the "=" command has not been used to arm the goto as a "motor address reassignment" command, then the software will:

- Wait for the prior "pending" x, y 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 the "*" acknowledgement character

For example,

```
*1000X
*-25687Y
*g
```

Would:

1. Set the next X value to 1000
2. Set the next Y value to -25687
3. Queue a GOTO on motor location 1000, -25687

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

On the other hand, if the "=" command has been used to "arm" the "G" command to reset the motor location (instead of doing a real goto), then this command will wait for the motor to go completely idle, and then will reset the motor location to match the current X,Y parameter values. Once this has been completed, the code will reset the 'G' mode back to "GoTo", so that the next "G" command will actually do a goto. As a simple example here,

```
*2000X
*1000Y
*2=
*g
```

This would redefine the current location to be X=2000, Y=1000, with no real motion occurring. See the "=" command for more information about this special one-shot mode.

H – Specify Motor Current When Moving

"H" is used to specify the motor current to use when motor motion is under way. A separate current may be specified for each motor. At power-on (or reset), the "H" value is set to **220** (which is the normal 1.3 amps of full power for the board), unless there is a jumper in the S1K position. In that case, the "H" value is set to **55**, which sets the windings to 0.3 amps.

The 'H' value is bit-encoded, with the low 8 bits (values 0-255) defining the actual current to use. The next 2 bits (bits 8 and 9) are used to block updating the X and Y motor currents with the new value, thus leaving the value unchanged. The actual bit encoding is:

Bit(s)	Numeric Value	Description
0-7	0-255	Actual current level being specified: values 23-255 are to be used for normal current levels (as shown below), a value of 0 is to be used to disable the motor completely
8	256	Block writing this value to the "X" motor (i.e., only write to the "Y" motor)
9	512	Block writing this value to the "Y" motor (i.e., only write to the "X" motor)

This means that you will be writing your current selection value to both the X and Y motors whenever you request a value of 0 to 255. If you just want to write to the "X" motor, add 512 to the current. If you just want to write to the "Y" motor, add 256 to the current. (Note that if you add both 512 and 256 to the value – i.e., 768 – the net effect is that neither motor current will be written.)

The ability of being able to specify which motor receives the current request is a factory-order option. By default, bits 8 and 9 are forced to 0 by the firmware, so that both motors are updated with the current request. As a factory option, this behavior can be changed to allow separately specifying the currents for each motor; however, if this option is selected, then you lose the ability to control the IO6 and IO7 TTL output lines.

For example, to write a current value of "100" to both the X and Y motors, issue the command:

100H

To request a current value of 100 for the "X" motor, and 50 for the "Y" motor, issue two commands as follows:

612H (i.e., 512 (to block 'Y') + 100 (current), to set X motor)

306H (i.e., 256 (to block 'X') + 50 (current), to set Y motor)

Use a current value of 0 to fully disable a motor.

Current values that are below 23 are not reliable; this is what specifies the minimum current that a given version of our board can generate.

The actual formula for determining the current delivered to the motor is

H = I*169

Thus, some commonly used values would be:

Current	***H value***
0.25	**43**
0.32	**55**
0.5	**84**
0.75	**127**
1.0	**169**
1.3	**220**
1.5	**255**

Note that the actual current delivered is probably no more accurate than 10-20%.

Please note that the board is only rated for continuous duty at the 1.3 amp level. The 1.5 amp setting is fine for running motors for short durations of time (less than 30 seconds), with at least 30 seconds of idle time (current to the windings off) in between motor motions.

I – Wait for motor 'Idle'

This allows your code to 'wait' for the motors to be idle. It also immediately reports what main type of action is occurring; this allows your code to confirm whether a new command can be sent. 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.

Most commands are executed immediately, and are not affected by new incoming data. However, both "G" (goto) and "A" (draw arc) have the restriction of not being able to queue more than 2 vectors (the one being drawn, and one which is pending). If a character is received while either of those commands are waiting for "room" to queue their next vector request, then the command ("G" or "A") which is waiting to be queued will be aborted. Neither of those commands will send the "*" (ready for new command) character until they have queued their last request; however, in a multi-board environment, your application may miss the "*". The "I"dle wait command gives you a method of safely resynchronizing with the board – it immediately sends back the type of wait which it is doing (as a single character response after the optional CR/LF command acknowledgement), and then will send the "*" when the command completes or is queued, as is appropriate.

The code immediately sends back one of 3 characters to inform you about the pending board operation:

- '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
- 'I' – the board has no Arc or unenqueued GoTo pending

After sending the above status character, the code then waits as is appropriate (depending on whether the unit is waiting on enquing an 'A' or 'G' command), and then sends an '*' response.

- Waiting on 'A': The '*' is sent as soon as the last vector of the arc is queued (motor motion is still occurring)
- Waiting on 'G': The '*' is sent as soon as the 'G' is queued (motor motion is still occurring)
- Waiting on 'I'dle: The '*' is sent once all motor motion has completed; the motors are stopped.

If you send ***any*** new command other than 'i' during the 'A' or 'G' styles of idle wait, then the 'A' will be aborted (i.e., the currently enqueued vectors will execute, but no further ones will be added to the list), and the 'G' will be discarded.

Additionally, if the wait is stopped by receipt of a new character, then the new character IS processed as part of a new command – it is NOT discarded.

For example, to go to a given X, Y location, and then wait completely for the motor to actually get there, you could simply issue the command sequence:

Send	Receive
1000X	*
3000Y	*
G	* *(note that the '*' is received as soon as the new X, Y location is actually accepted as the next item which will be targeted)*
I	I* *(note that this '*' is not received until all motion completes)*

If you already had multiple "G" requests queued, and you issued the "I" before the "G" was able to queue its request, then you might have received a "G" instead of an "I" in response to the "I" request, above. This would mean that "G" is still trying to queue its request, and that it is not safe to send any other character except 'I'. The "G*" response would be sent if the board has just enqueued the G command. The "I*" response would be sent if the board is truly idle.

K –Set the "Stop oK" rate

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

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

For example,

100k

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

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

62500K

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

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.

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

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

The latched events reported are as follows:

Bit	Value	Description
0	+1	Y- limit reached during a Y- step action
1	+2	Y+ limit reached during a Y+ step action
2	+4	X- limit reached during a X- step action
3	+8	X+ limit reached during a X+ step action
4	+16	System power-on or reset ("!") has occurred
5	+32	'G' or 'A' command was probably aborted – a non "~" or "I" character was received while the G or A was waiting for queue space to save a vertex location.

For example, after initial power on,

```
L
```

Would report

```
L16
*
```

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

```
L4
*
```

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): This mode may only be used if the motor winding current setting (the 'h' and 'w' commands) does not exceed 26. In some (rare) cases it may increase the top speed of the motor; usually, due to the way that current is regulated in the controller, it will actually decrease the top speed!
- **3 : Microstep, as fine as 1/16th 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 **1/16**th 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 8 microsteps. The "full step" location will be the (microstep location)/16, with fractions of 0 to 7/16 mapping into fractional location 0, and 8/16 though 15/16 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 16 microsteps. The "full step" location will be the (microstep location)/8, with the fractional part forced to 0.5. ***This mode may only be used if the motor winding current setting (the 'h' and 'w' commands) does not exceed 26.*** In some (rare) cases it may increase the top speed of the motor; usually, due to the way that current is regulated in the controller, it will actually decrease the top speed!
- 3: Microstep mode. The current through the windings are precision-controlled, so that the microposition can be obtained. The physical motor position expressed in full step units is the (microstep location/16).

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 44801 microsteps per second per second. If a value of 0 is specified, the code forces it to have a value of **80**00. If a value above 44801 (or less than 0) is specified, the code will accept it, but will ramp unreliably (i.e., do not do it!).

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

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

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.

R – Set run Rate target speed for the faster motor

This defines the run-rate to be used for the faster motor. It may be specified to be between 1 and 44801 microsteps per second. If a value of 0 is specified, the code forces a value of **800**. If a value outside of the limits is specified, then it is accepted, but the code will not operate reliably. As with the ramP rate, do not specify values outside of the 1-44801 legal domain.

This defines the equivalent number of microsteps/second which are to be used to run the faster motor under the GoTo or Arc command. The internal motor position is updated at this rate, using a sampling interval of 44801 update tests per second. The motor windings are then updated according to the stepping mode. For example, if the stepping mode (the 'o' command) for a given motor is one of the full-step modes instead of the microstep mode, and the microstep resolution is set to '1', then the motor will actually experience motion at **1/16**th of the specified rate.

For example,

250R

Sets the stepping rate to 250 microsteps per second.

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

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 'R' command is forgotten.

V – Verbose mode command synchronization

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

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

"W" is used to specify the motor current to use when the motor is idle. A separate current may be specified for each motor. At power-on (or reset), the "W" value is set to 0, which disables the windings when motion is complete.

As with 'H', the 'W' value is bit-encoded, with the low 8 bits (values 0-255) defining the actual current to use. The next 2 bits (bits 8 and 9) are used to block updating the X and Y motor currents with the new value, thus leaving the value unchanged. The actual bit encoding is:

Bit(s)	Numeric Value	Description
0-7	0-255	Actual current level being specified: values 23-220 are to be used for normal current levels (as shown below), a value of 0 is to be used to disable the motor completely
8	256	Block writing this value to the "X" motor (i.e., only write to the "Y" motor)
9	512	Block writing this value to the "Y" motor (i.e., only write to the "X" motor)

This means that you will be writing your current selection value to both the X and Y motors whenever you request a value of 0 to 255. If you just want to write to the "X" motor, add 512 to the current. If you just want to write to the "Y" motor, add 256 to the current. (Note that if you add both 512 and 256 to the value – i.e., 768 – the net effect is that neither motor current will be written.)

The ability of being able to specify which motor receives the current request is a factory-order option. By default, bits 8 and 9 are forced to 0 by the firmware, so that both motors are updated with the current request. As a factory option, this behavior can be changed to allow separately specifying the currents for each motor; however, if this option is selected, then you lose the ability to control the IO6 and IO7 TTL output lines.

For example, to write an 'idle current' value of "100" to both the X and Y motors, issue the command:

100W

To request an 'idle current' value of 100 for the "X" motor, and 50 for the "Y" motor, issue two commands as follows:

612W (i.e., 512 (to block 'Y') + 100 (current), to set X motor)

306W (i.e., 256 (to block 'X') + 50 (current), to set Y motor)

If you want to turn off power to the windings when motion is complete, set "W" to 0.

Values for W that are below 23 are not reliable, although values down to 12 will usually be good enough for use with the 'W' command. This is what specifies the minimum current that a given version of our board can generate. For values below 23, the actual current provided will be significantly above that shown in the following table. Lower values of W will have larger errors.

The actual formula for determining the current delivered to the motor is

H = I*169

Thus, some commonly used values would be:

Current	*H value*
0.25	**43**
0.32	**55**
0.5	**84**
0.75	**127**
1.0	**169**
1.3	**220**
1.5	**255**

Note that the actual current delivered is probably no more accurate than 10-20%.

Please note that the board is only rated for continuous duty at the 1.3 amp level. The 1.5 amp setting is fine for running motors for short durations of time (less than 30 seconds), with at least 30 seconds of idle time (current to the windings off) in between motor motions.

X – Set next "X" value as defined by the current "=" mode

This command sets the next X parameter to the value requested, as defined by the current "=" mode. By default, this is a direct assignment of the value specified to the pending X register.

For example,

100X

Would normally set the next X value to be 100.

However, if the "=" command has been used to reset the default assignment mode from absolute to relative, then each "X" command will simply do a signed addition of the new value to the prior X, Y request value. For example

100X200X

would start by setting X to 100 (assuming it 'really' started at 0), and then would add 200 to that, thus making the current X be 300.

Y – Set next "Y" value as defined by the current "=" mode

This command sets the next Y parameter to the value requested, as defined by the current "=" mode. By default, this is a direct assignment of the value specified to the pending Y register.

For example,

100Y

Would normally set the next Y value to be 100.

However, if the "=" command has been used to reset the default assignment mode from absolute to relative, then each "Y" command will simply do a signed addition of the new value to the prior X, Y request value. For example

100Y-200Y

would start by setting Y to 100 (assuming it 'really' started at 0), and then would add -200 to that, thus making the current Y be -100.

Z – Stop motors.

'Z' 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".

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

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

The value passed is used as the new microstep size, in fixed $1/16^{th}$ of a full step units. **At raw power on, the board acts like a "1!" has been requested**; that is to say, it sets the microstep size to 1x1/16, 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 16; it is clipped to this range if exceeded.

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

For example,

1!

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

The reset command also selects the following settings:

- **2=** – Select the next "G"oto assigns the location instead of moving the motors
- **0X** – Set next X will be 0
- **0Y** – Set next Y will be 0
- **G** – Reset both motors to be at location 0
- **220H** – Set motors to full power mode (note: '55H' is emulated if the ¼ power mode hardware strap is set via installation of the S1K jumper
- **80K** – Set the "Stop OK" rate to **80** microsteps/second
- **3O** – Set the motor windings Order to "microstep"
- **8000P** – Set the rate of changing the motor speed to **80**00 microsteps/second/second
- **800R** – Set the target run rate for the faster motor to 800 microsteps/second
- **1V** – Set <CR><LF> sent at start of new command, no transmission delay time
- **0W** – Windings are off when idle

= – *Define interpretation of "X", "Y", and "G" commands*

This command is used to define how the X and Y commands operate (that is to say, whether they directly assign values to the X and Y parameter variables, or they add new offsets to those values), and to define whether the "G" command actual operates as a "GoTo" or as a one-shot "assign location" command.

It is bit encoded as follows:

Bit	Use
0	0 = Assign X, Y to value requested
	1 = Add value requested to X, Y (i.e., 1000X would add 1000 to the current pending X value)
1	0 = "G" queues the current X, Y parameters as a new GoTo target
	1 = The next "G" waits for all motor motion to stop, then assigns the current motor location from the current X, Y values. Note that "G" then automatically clears this bit after performing the assign

Therefore, the legal values become:

Value	Use
0	Default: "X" and "Y" commands are absolute, "G" is a "GOTO"
1	"X" and "Y" commands are relative to their current values, "G" is a "GOTO"
2	"X" and "Y" commands are absolute, the next "G" command is an "ASSIGN MOTOR LOCATION"
3	"X" and "Y" commands are relative to their current values, the next "G" command is an "ASSIGN MOTOR LOCATION"

As an annotated example (the '*' are sent by the controller),

```
*2=     - Set the X and Y parameter mode to absolute, next G assign location
*0X     - Set current X and
*0Y     - Y parameters to 0
*G      - Since we were in "Assign motor location" mode, the current location
                is now 0,0; no actual motor motion occurs
*1=     - Set the X and Y parameter mode to relative
*200X   - X parameter is now 0+200➔200
*-300Y  - Y parameter is now 0-300➔-300
*G      - go to location 200,-300
*-100X  - X parameter is now 200-100➔100
*200Y   - Y parameter is now –300+200➔-100
*G      - go to location 100,-100
```

? – 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:

- 5: Port A register – this contains the limit switches
- 6: Port B register – this contains the TTL inputs
- 7: Port C register – this controls the motor windings, and it contains the two extra bits which are used as the 'IO6' and 'IO7' signals on the board

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

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

- -1; Report current X location
- -2; Report current Y location
- -3; Report target X location
- -4; Report target Y location
- -12; Report current software version and copyright
- other: Treat as 0 (report all except version/copyright)

All of the reports follow a common format, of:

1. If Verbose Mode is on, then a <carriage return><line feed> ("crlf") pair is sent.
2. 'R' is sent.
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 (or set of values) is (are) reported.
7. If Verbose Mode is on, then a <crlf> is sent

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

The special reports which are understood are as follows:

0: Report all reportable items

The "report all reportable items" mode reports the data as a comma separated list of values, for reports –1 through –4. Just after power on, for example, the request of "0?" would generate the report:

R,0,a,b,c,d

Where:

- R is the "Report" flag character
- 0 is the report number; 0 is the 'all' report
- a is the value for the current X location (report "-1")
- b is the value for the current Y location (report "-2")
- c is the value for the target X location (report "-3")
- d is the value for the target Y location (report "-4")

For example,

```
B0?
```

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

```
X,0,30,10,1000,30,10
*
```

-1: Report current X location

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

For example,

```
-1?
```

You could receive:

```
R,-1,10
*
```

-2: Report current Y location

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

For example,

```
-2?
```

You could receive:

```
R,-2,2502
*
```

-3: Report the target X location

This reports the current GoTo X target location.

For example,

```
-3?
```

You could receive:

```
R,-3,10
*
```

-4: Report the target Y location

This reports current GoTo Y target location.

For example,

```
-4?
```

You could receive:

```
R,-4,-35443
*
```

-12: Report current software version and copyright

This reports the software version and copyright.

For example,

```
-12?
```

could report:

```
BC2DNCStepper.src $version: 1.29$
©2002 by Peter Norberg Consulting, Inc. All Rights Reserved
*
```

Any character above 'z' – stop sending pending output data, then skip

Characters above 'z' (such as '{', '~', and '}') are actively ignored by the processing code for most actions. Any pending output (such as the copyright message or any prior responses) will be cancelled, the current numeric value will normally be treated as completed, and the character will otherwise be fully ignored.

We suggest that the '~' character be used by applications which are not maintaining a perfect synch with the board as a "clear output buffers". For example: Send the sequence

~~I

Then start to monitor the incoming data stream for new serial characters received since transmission of the "I" (i.e., discard all data received prior to the 'I' being transmitted). That data will strictly be the response to the 'I' (idle wait) command; there will be no "left-overs" from prior commands.

other – stop sending output data, then echo the '' command complete response*

Any illegal command is normally ignored, other than sending a response of "*". However, if a numeric input was under way, that value will be treated as complete. If an "A" or "G" command still has pending actions waiting to be queued, the queued actions are aborted.

For example,

123 456X

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

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

SerTest.exe – Command line control of stepper motors

The SerTest.exe application (provided as part of the sample software) is a simple tool which allows command line based control of the BC2DNCStepper product line (the SimStep and BiStep boards). It allows a batch-based script to control stepper motors, with no further need for any programming knowledge. All sources are provided, to allow rewriting as needed.

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

```
SerTest Text1 Text2 … Textn
```

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

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

  For example,

  /b9600 sets 9600 baud,

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

- /c??? Set the extended list of "long wait" commands

  This is used to set the list of commands which may take a long time to respond. By default, just "I" (for "Idle Wait") is specified to take more than $1/10^{th}$ of a second; however, when operating the BC2DNCStepper firmware, the commands "A" (for "draw Arc") and "G" (for "Go draw the current line) can take a large amount of time, if no queue space is left for saving the new requests. When using the BC2DNCStepper firmware, you therefore should include the switch:

  /cag

  in order to add the "A" and "G" commands to the idle wait time list.

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

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

  /i60000 – Set 1 minute before timeout

  /i10000 – set 10 seconds before timeout

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

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

  SerTest /pCOM2 4!x1000g

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

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

For example,

Sertest /cag 1!1000x-2000ygi

Would:

1. Operate at 9600 baud on COM1 using a 1 minute time out
2. reset the board to operate with a microstep size of **1/16**
3. tell the X motor to go to location 1000,
4. tell the y motor to go to location -2000,
5. and wait up to 60 seconds for the motions to complete

Similarly,

SerTest /cag /pCOM2 /b2400 /i10000 5000yg

Would:

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

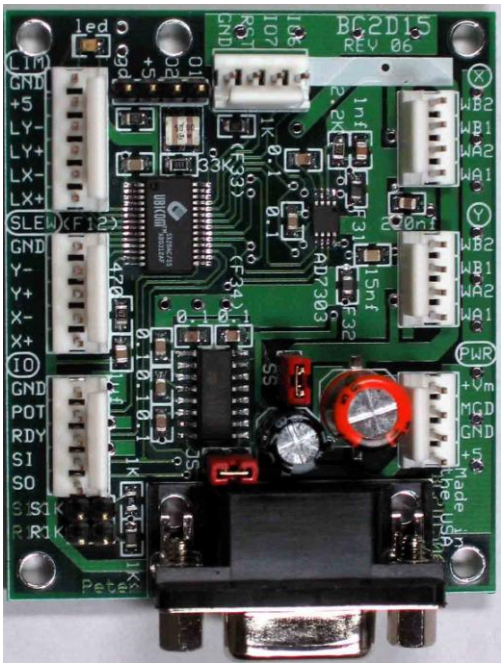
StepperBoard.dll – An ActiveX controller for StepperBoard products

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

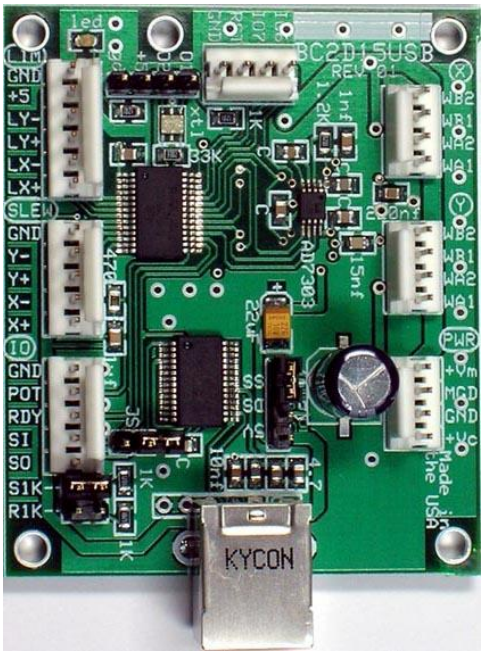
The program is well-documented in the manual StepperBoardClass.pdf. Please refer to that manual for more information about the product.

Board Connections

Examples of the BC2D15 series of boards with the MTA-100 connector option are shown below.



BC2D1506



BC2D15USB

Board Size

The boards, oriented as shown on this page, are 2.5 inches high by 2.1 inches wide.

Mounting Requirements

The BC2D15 mounting holes are 0.125 inches in diameter, and are positioned exactly 0.125 inches in from each corner. They allow up to a number 5 screw, which thus allows use of the standard #4 mounting spacers. Vertically, their centers are 2.25 inches apart, and horizontally they are 1.85 inches apart. Thus, when the board is positioned as shown above, their positions are:

(0.125, 0.125), (1.975, 0.125),
(0.125, 2.375), (1.975, 2.375)

Connector Signal Pinouts

There are nine connectors on each board.

Going from top-left down, we have:

- Extra TTL logic connector (GND, RST, IO7, IO6)
- SX-Key debugger connector (4 pin SIP header)
- SX-28 Direct Access signals:
  - TTL Limit Input (GND, +5V, LY- to LX+)
  - TTL Motor Direction Slew Control (Y- to X+)
  - Board status and TTL Serial (NX, RDY, SI (serial input), SO (serial output)). Only use the TTL serial if the JS jumper, located near the bottom-left portion of the board, is removed.

Then, on the bottom we have:

- RS232 Serial, as a DB9 female connector on the bottom of each board. There is also an option to replace the connector with a 4 pin SIP or MTA-100 header.

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

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

Extra TTL Logic Connector

Pin	Name	Description
1	GND	Vss (gnd)
2	RST	Reset board when pulled low
3	IO7	IO7 extended TTL signal
4	IO6	IO6 extended TTL signal

This connector allows access to the extended TTL I/O signals IO6 and IO7, as well as the RESET signal. See the 'C', 'T', and the '7?' commands for access to the IO6 and IO7 lines.

SX-Key debugger connector

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

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

TTL Limit Input and Reset

Name	Description
GND	Ground reference for inputs – short input to GND to denote limit
+5	+5 available for external use; please draw less than 0.25 amps
LY-	Y Minimum limit reached, when low
LY+	Y Maximum limit reached, when low
LX-	X Minimum limit reached, when low
LX+	X Maximum limit reached, when low

The LIM connector is used to warn the SX-28 that a limit is being reached in the given direction. The LX and LY signals are designed to be shorted to GND to denote that a limit is present; they are also compatible with normal TTL outputs from any TTL compatible device. LY- through LX+ are internally pulled up to +5 with 10-33K resistors (within the SX-28 itself).

TTL Motor Direction Slew Control

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

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

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

Board status and TTL Serial

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

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

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

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

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

RS232 Serial DB9 Female (socket), for the RS232 product versions

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

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

The SIP header option is fully labeled for access to the same signals, and also includes access to the board's +5 supply.

USB Serial (BC2D15USB)

Name	Description
SHD	USB Shield
GND	USB Signal Ground (USB pin 4)
DP3	USB Data signal plus (USB pin 3)
DM2	USB Data signal minus (USB pin 2)
U5V	USB +5V (USB pin 1)

Normally, this connector is installed as a normal USB-B connector. As an order option, a SIP header may be installed, so that you can connect the system to an external USB plug/

This connector provides for all external serial communications, using the USB standard. Note that the portion of the board which supports the USB communications is powered by the USB +5V and USB Signal Ground pins; that is to say, as soon as the USB cable is connected between the board and the computer, and power is applied to the cable, then the USB portion of the BC2D15USB is enabled. This does not include enabling of the rest of the board, however! All that is enabled is that the computer can recognize that there is a USB device present.

Power Connector

The power connector is labeled differently depending on the board artwork revision, and has different associated power options.

Powering the BC2D15 artwork version 06 (original release)

The power connector for the original BC2D15 is as follows:

Name	***Description***
+Vm	7.5-34 volts, for the X and Y motors
MGD	Ground for Vm
GND	Ground for Vc
+5	+5 volts for the logic circuits. This must be provided by your supply if the 'SS' jumper is removed, it is available for use if the 'SS' jumper is installed

There are several ways of powering the system, which are dependent upon the current and voltage requirements of the system and on the board version. One or two power supplies may be used, depending upon the voltage needed by the motors and upon whether extra cooling can be applied to the voltage regulator. If the motors require more than 19 volts to operate, **using the same power supply to the regulator will cause the regulator to get extremely hot** (over 100 deg. C). Although it technically can withstand temperatures up to 150 deg. C, we do not recommend or warrant it. It is much better in this case to split the supplies. Use pins 3 (GND) and 4 (+5) to provide 5 volts to the board (up to 1 amp, if a 5 volt fan is being used). Use pins 2 (MGD) and 1 (+Vm) to provide the motor power in this case, and ***REMOVE THE 'SS' JUMPER FROM THE BOARD. OTHERWISE YOU WILL BE CONNECTING YOUR 5 VOLT POWER SUPPLY TO THE OUTPUT OF OUR 5 VOLT REGULATOR!*** The BC2D15 has a single jumper (labeled 'SS') that gives you two options on powering the system.

The power options for the original BC2D15 can be summarized as:

Motor Voltage	***SS Jumper Installed***	***Use sep. Power supply***	***Comments***
6-19V	YES	NO	Single power supply, connected to pins 2 (MGD) and 1 (Vm) of power connector. SS Jumper is installed.
>19V	NO	YES	Use two power supplies, one for the motor (connected to pins 2 (MGD) and 1 (Vm)), the other for the digital power (pins 3 (GND) and 4 (+5v)). The digital power should be 5 volts, at least 300 ma if no fan is used, and 1 amp if a fan is used. The SS jumper is removed.

Powering the BC2D15 artwork version 09 (ROHS release) or the BC2D15USB

The power connector for the revised versions of the BC2D15 is as follows:

Name	***Description***
+Vm	7.5-34 volts, for the X and Y motors
MGD	Ground for Vm
GND	Ground for Vc
+Vc	Logic supply voltage, value dependant on jumper options. This will be 5 volts (jumper set to '5V') , 6.5 to 15 volts (jumper set to to 'DS'), or not connected (jumper set to 'SS')

There are several ways of powering the system, which are dependent upon the current and voltage requirements of the system and on the board version. One or two power supplies may be used, depending upon the voltage needed by the motors and upon whether extra cooling can be applied to the voltage regulator. Please note that if the motors require more than 19 volts to operate, **using the same power supply to the regulator will cause the regulator to get extremely hot** (over 100 deg. C). Although it technically can withstand temperatures up to 150 deg. C, we do not recommend or warrant it. It is much better in this case to split the supplies.

If a dual power supply configuration is selected, use GND and +Vc to provide either 5 volts (set the power jumper to "5V") or 6.5 to 15 volts (power jumper set to "DS") to the board (up to 0.3 amp, if our 5 volt fan is being used). Use MGD and +Vm to provide the motor power. Use regulated DC supplies for both voltage sources.

If a single power supply configuration is selected, use MGD and +Vm to power the board and motors. The supply's voltage must be regulated DC, in the range of 6.5 to 15 volts.

MAKE CERTAIN THAT THE POWER JUMPER CORRECTLY MATCHES YOUR POWER SELECTION – '5V' FOR A 5 VOLT LOGIC SUPPLY, OR 'DS' FOR A 6.5 TO 15 VOLT SUPPPY. Failure to do so will destroy the board!

The power options can thus be summarized as:

Motor Voltage	**Power Jumper Setting**	**Use sep. Power supply**	**Comments**
6.5-15V	SS	NO	Single power supply, connected to pins MGD and +Vm of power connector. The SS jumper is installed on the power options header
15-26V, logic supply is 6.5 to 15 volts	DS	YES	Use two power supplies, one for the motor (connected to pins MGD and +Vm), the other for the digital power (pins GND and +Vc). The digital power should be 6.5 to 15 volts. The DS jumper installed is installed on the power options header
15-26V, logic supply is 5 volts	5V	YES	Use two power supplies, one for the motor (connected to pins GND and +Vm), the other for the digital power (pins GND and +Vc). The digital power must be 5.0 volts. The 5V jumper installed on the power options header.

When the system is jumpered for the '5V' setting, the on-board power regulator is fully bypassed. You must provide exactly 5.0 volts to the +Vc/GND pins, in order to correctly operate the board***. If this voltage ever exceeds 6 volts, you will destroy the board!*** If it ever drops to 4.2 volts or less, the board will undergo a reset, and will act as if power has just been applied (losing all settings).

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)	1.7	2.3

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

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

For example, if you are going to microstep (mode 3) a motor whose winding current has been calculated to be 0.4 amps, then your power supply needs to be able to supply 2.3 x 0.4, or 0.9 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 7.5 volt power supply, therefore you would us at least a 7.5 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 BC2D15 series requires 300 mA for operation if the optional fan is installed. It requires 200 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 7.5 to 15 volts, then you may choose to use a single supply to operate the system. Position the power jumper at the "SS" location, and connect the power to Vm and GND.

Dual Supply

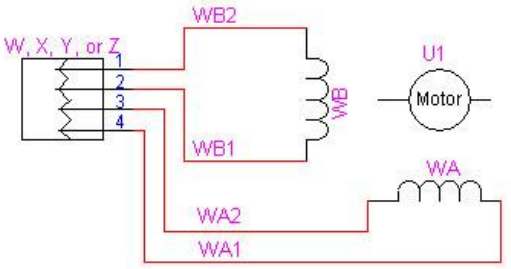
You may separate the motor supply from the logic supply. If you do so, you will need to provide the 5 volts for the logic system.

The motor supply should be above 7.5 volts in all cases (due to some signal requirements on the board), and otherwise is as calculated under sections 1 through 3, above. If the supply is to drive 2 motors, please remember to double the current needs.

ALWAYS MAKE CERTAIN THAT THE POWER OPTION JUMPER (SS, DS, 5V) MATCHES YOUR POWER SUPPLY ATTACHMENT TECHNIQUE!

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, BC4D15 and the BiStep2A series of controllers (pins counting from top to bottom):



Typical Bipolar Motor Connection
To the BC2D15 or BC4D15 Board

Pin	Name	Description
1	WB2	Winding B, pin 2
2	WB1	Winding B, pin 1
3	WA2	Winding A, pin 2
4	WA1	Winding A, pin 1

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 motor, number the leads (from 1 to 4 for a bipolar motor, from 1 to 5 or 6 for a unipolar motor). Then measure the resistances and record the values in the empty cells in a table like the following: (IF YOUR MOTOR IS UNIPOLAR, PLEASE CONTACT US. WE CAN USE THE BC4D15 TO OPERATE SOME UNIPOLAR MOTORS IN BIPOLAR MODE!)

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

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

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

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

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

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

You can often operate a 6-wire unipolar motor as if it were a 4-wire bipolar motor (when using the BC2D15 or BC4D15 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 correct "H" command to select the correct motor winding current. For example, if your motor requires 0.5 amps/winding, issue the command

  84h

- Issue the command "800rx1000gi0gi", which should cause the motor to spin to logical location 1000, then back to 0.
- 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

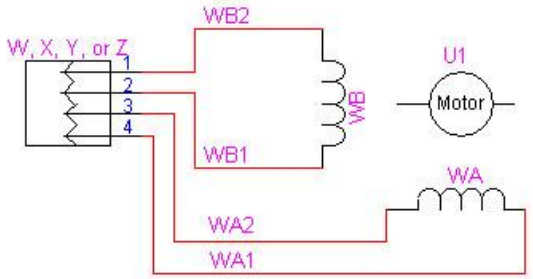
B A D C.

Motor Wiring Examples

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

Bipolar Motors

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



Typical Bipolar Motor Connection
To the BC2D15 or BC4D15 Board

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

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

The wiring of this unit is:

Color	BC4D15
Yellow	WB2
Black	WB1
Red	WA2
Gray	WA1

The "H" command to be issued after any power-on or reset action in order to select the correct current for motor operation is:

135h

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

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

Color	BC4D15
Brown	WB2
Orange	WB1
Yellow	WA2
Red	WA1

The "H" command to be issued after any power-on or reset action in order to select the correct current for motor operation is:

68h

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

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

Color	BC4D15
Yellow	WB2
White	WB1
Blue	WA2
Red	WA1

The "H" command to be issued after any power-on or reset action in order to select the correct current for motor operation is:

47h

Jameco 168831 12 Volt, 1.25 Amp

This motor is a Superior Electric "SLO-SYN" stepping motor, model number SM-200-0050-HL. We tested it with the wiring of:

Color	BC4D15
White/Brown	WB2
Brown	WB1
White/Yellow	WA2
Yellow	WA1

The "H" command to be issued after any power-on or reset action in order to select the correct current for motor operation is:

211h

Install The Fan Assembly

When the BC2D15 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 BC2D15 via two 1-inch stand-offs, and is powered off of a 4-pin SIP header that is available on the board.

1. Attach the two 1-inch stand-offs to the BC2D15. Use two of the split-ring washers and the two nuts to attach the stand-offs; they are positioned as shown in the picture on the following page labeled "Unit with stand-offs".
2. Connect the fan power to the power portion of the 4-pin SIP header at the top left corner of the board (this will be under the fan once it is installed). The black lead goes to the left-most pin (gnd), the red lead goes to the next pin on the header (+5). The two right-most pins of the board SIP header are not used during normal operation of the board.
3. Attach the fan to the stand-offs using the remaining two screws and lockwashers. **The label side of the fan must face down** as shown in the photo on the next page, labeled "Unit with Fan". ***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 is not covered by the warranty.***

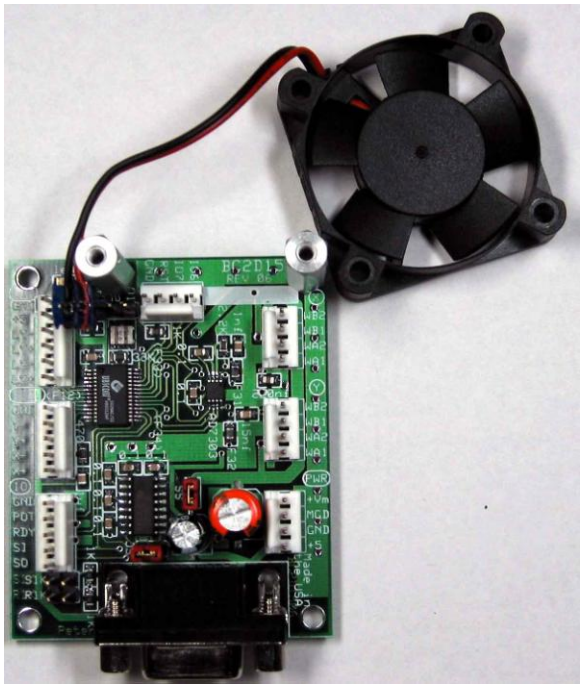


Figure 1: Unit with Mounting Posts

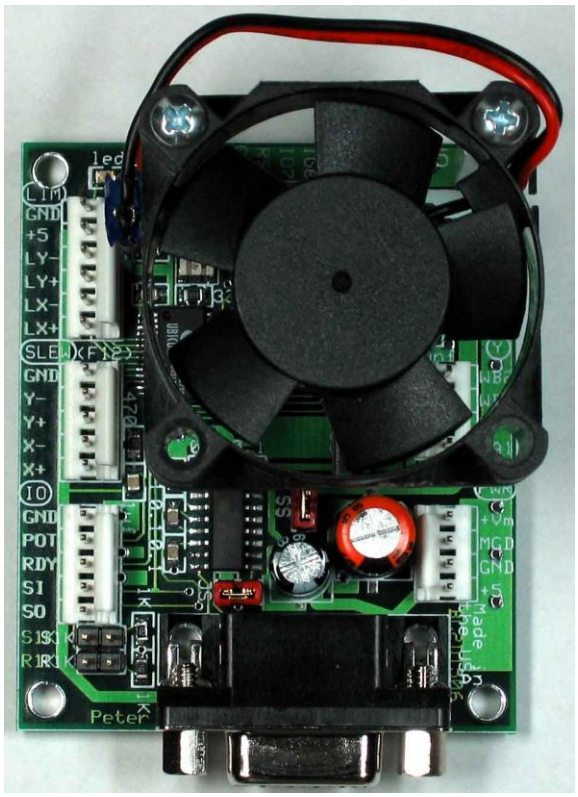


Figure 2: Unit with Fans

Please note that the direction of air flow is "down", towards the board. If the fans are mounted such that the air flow is "up" (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 fan blows air down onto the top of the board***. If either action does not occur, turn the system off, and trace the problem.

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