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

Information and Instruction Manual for SD4DFifoNCStepper Firmware and the SD4DP series of Stepper Motor Controllers

By
Peter Norberg Consulting, Inc.
Matches SD4DFifoNCStepper Firmware Revision 4.1

Authored in the United States of America. Manual published December 10, 2012 4:19 PM

Table Of Contents

Disclaimer and Revision History

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

This manual describes artwork SD4DP. The firmware release described is SD4DFifoNCStepper version 3.5. The manual version shown on the front page normally has the same value as the associated SD4DFifoNCStepper version. If no manual has yet been published which matches a given firmware level, then the update is purely one of internal details; no new command features will have been added.

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

Version	Date	Description
1.1	May 15, 2007	First manual release
1.2	June 6, 2007	Added new bit to 'O' command to enable use of 'step-per-direction' mode controllers (controllers which use a separate pulse input per direction of travel)
3.1	July 24, 2007	Changed version number to match SD4DNCRouter, to reflect new command synchronization protocol.
3.5	December 12, 2007	Corrected issue with binary data commands, which would cause null data bytes to be dropped
	May 16, 2008	Correct documentation error on R1K/S1K: usage was flipped in summary section
	June 5, 2008	Adjusted hardware notes for the new ESD protected version of the SD4DP board (serial number M209-0001 and above).
4.0	March 8, 2009	Corrected error in chain motion that could cause unexpected motor target location if the 'WXYZ' mode of target selection is performed. Added hex data report mode Added 'I' command for latched IO port data Changed WXYZ specification to also include selection of the motor as the target for reports and backlash actions (i.e., 'M' selection implied)
	January 10, 2010	Added missing documentation for function introduced in version 3.6 (instant-stop via TTL input IO5)
4.1	June 12, 2011	Added control of lead-time into step pulse, to satisfy settling time requirements for first step after a direction change on one controller
	October 5, 2011	Added caveat about step sizes supported on the external step-and-direction driver boards.

Product Safety Warnings

The SD4D series of controllers can get hot enough to burn skin if touched, depending on the voltages and currents used in a given application. Care must always be taken when handling the product to avoid touching the board or its installed components, until the board has cooled down completely. Always allow adequate time for the board to "cool down" after use, and fully disconnect it from any power supply before handling it.

The board itself must not be placed near any flammable item, as it can generate significant heat. There exist several components on the bottom side of the board that can get quite hot; therefore, the board must be correctly mounted using stand-offs.

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

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

LIFE SUPPORT POLICY

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

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

Introduction and Product Summary

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

The **SD4D** stepping motor controller from Peter Norberg Consulting, Inc., has the following general performance specifications:

	SD4DP
Logic supply voltage (+5V)	5V or 6.5 to 15V
Average Current draw	150 mA; 1 A if all drivers are active and driving a significant load
Board size	3" x 1.6"
ULN2803 Driver for Gecko drive compatibility	Yes

The SD4DFifoNCStepper firmware is designed to allow simultaneous linked control of up to 4 step-and-direction driver boards (such as those produced by Applied Motion or Gecko) using the SD4D controller. Each board is operated via the USB serial interface, which provides full access to the controller's extreme range of stepping rates (1 to 60,000 steps per second), slope rates (1 to 60,000 steps per second per second), and various motor motion rules are provided. The boards permit setting of the 'polarity' of the step and direction pulses, so that more step-and-direction board drivers may be controlled.

Due to timing limitations of the ramping algorithm that is used to drive the 'slower' motors in a multi-axis environment, 1/2 step and full step driver boards may not work correctly with all combinations of ramps and rates. The slower motors are always stepped at the same time as the fastest motor, with the slower rates being defined by effectively 'missed' steps. For large step sizes (1/2 and full, and possibly some 1/4 as well) this can cause a motor that is near its resonance frequency to stall, if this skipped-step timing happens to match the resonance frequency of the motor.

It is strongly recommended that 1/8 step and finer resolution step-and-direction driver boards be used with this product. Such finer microstepping bypasses the issues induced by the modulus arithmetic in the ramping algorithm.

The SD4DNCStepper firmware shares many of the features of the SD4DPotRouter four-motor controller firmware. The step and direction control of the motor drivers is identical, as is the general method of sending numeric parameters for commands. Many of the commands which configure the system are also identical (such as setting the step rate); however, the fundamental control theory is different. The SD4DNCStepper firmware explicitly controls all four motors at the same time, from a single command (such as Goto or Arc), with automatic step-rate ratioing in order to generate straight lines; while SD4DPotRouter explicitly controls the motors independently, so that one motor may be performing a "slew" operation, while another is executing a "goto".

The system operates by your first setting up the parameters (such as the next X, next Y, etc.), and then executing a command (such as "G", for "Go to the new X, Y location). As a simple annotated example, the commands given could be as follows (the '*' or 'G0*' character(s) is/are sent by the controller as a "ready" prompt; the rest are commands sent):

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

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

Short Feature Summary

- Up to four stepper motors may be controlled at one time via step-and-direction pulses.
- The controller may be configured to generate step-and-direction signals (compatible with most drivers, such as Gecko and Applied Motion) or step-per-direction signals (compatible with drivers which require a separate pulse source per direction of motor motion)
- Supports a separate backlash setting for each motor
- Limit switches may optionally be used to automatically request motion stop of any motor in either direction.
- Rates of 1 to 60,000 steps per second are supported.
- Step rates are changed by linearly ramping the rates. The rate of change is specified for the 'fastest' motor, and the code automatically scales the remaining rates to cause 'straigt-line' motion. It can be from 1 to 60,000 steps per second per second.
- All motor coordinates and rates are always expressed in logical step units. The actual step sizes that are used are dependent upon the motor driver boards which are controlled by the SD4D unit.
- Motor coordinates are maintained as 32 bit signed values, and thus have a range of -2,147,483,647 through +2,147,483,647.
- The code automatically scales motor motions between the four motors, so that the 'fastest' motor operates at the target rate which you specify, and so that all motors arrive at their target destinations at effectively the same time.
- Permits 'chaining' of goto-vectors, so that you don't have to stop between each line
- Complete control of the motors, including total monitoring of current conditions, is available through the high speed USB serial connection.
- Runs off of a single user-provided 6.5 to 15 volt regulated DC power supply, or a single user-provided 5 volt regulated DC power supply.
- An additional 7 generic TTL I/O lines are available for any required use.
- The SD4DG adds a driver which has adequate current capabilities to drive the Gecko series of stepper controllers.
- Automatic Latching edge detection of signal changes on IO ports IO0 through IO6.

Firmware Configuration

The SD4DFifoNCStepper firmware has a set of initial settings that are selected at power-on or reset that may be reconfigured at the time the product is ordered. With the exception of the serial baud rate used, all of these features may be reset through use of the appropriate serial command.

Default Stop Rate

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

Default Slew Rate

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

Default Ramp Rate

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

Default 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 I/O Port Directions

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

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

Default initial I/O Port Values

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

Default mode of control – step-and-direction or step-per-direction

By default, the controller is configured to generate step-and-direction signals. That is to say, a pair of outputs is generated per motor, one consisting of the step pulses, the other defining which direction the motor is to spin on each step.

As of firmware version 1.2 and later, the system now also supports the "step-per-direction" controllers through use of an extra bit in the 'O' command. In this configuration, the standard "step" outputs are used for generation of steps in the positive direction, while the standard "dir" outputs are used for generation of steps in the minus direction.

You may order the firmware preconfigured to operate in the "step-per-direction" mode, if this is needed.

Hardware Configuration: Board Jumpers

The SD4DFifoNCStepper firmware has three features that can be configured as startup options through use of a hardware strap.

Jumper R1K: Default Step Pulse Polarity

Normally, we ship the product such that the default operation of the STEP output pulses is OFF-ON-OFF; that is to say a low-going pulse (of user-programmable width) is generated for each step requested. If the R1K jumper is installed, then this pulse is ON-OFF-ON (high going). The 'O' command may be used to override the jumper setting.

Jumper S1K: Default Step Direction Level

Normally, we ship the product such that the default operation of the DIRECTION output signals is "OFF=Negative, ON=Positive". That is to say, when a pulse occurs on the associated step line, the direction line will be OFF if the step is to be negative, or ON if the step is to be positive.

If the S1K jumper is installed, then this operation is reversed: LOW becomes positive, and HIGH becomes negative. The 'O' command may be used to override the jumper setting.

Jumper DP: Select pullup power source

The DP jumper is used to determine the pullup voltage used on the step-and-direction outputs of the board.

When the jumper is positioned across the inner two pins of the 3 pin DP header, then the pullup voltage is 5 volts, which generates TTL-compatible outputs.

When the jumper is positioned across the outer two pins of the 3 pin DP header, the outputs are pulled up to the +Vm voltage. This position should be used when operating opto-isolated inputs that are configured for high voltage use (such as 12 or 24 volt systems), and the +Vm voltage must match that high voltage.

Cooling Requirements

Normally, the board does not require any special cooling. However, if you use the +5V power outputs from the board for driving your own circuits and if you are supplying 6.5 to 15 volts as your power source to the board, then you will want to fan-cool the board if you are going to be drawing more than about 100 mA of power.

You may also need to provide for board cooling if you are driving multiple outputs at 4 mA of current or more. If, in your application, the SX48 seems to be getting too warm, then you need to (1) check your connections to make certain that the 5mA of current/output requirement is not being exceeded, and (2) cool the board.

Fan based cooling should be done such that the airflow is directed toward the top or bottom surface of the board, centered over the SX48 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 (which cools the SX48 microprocessor better), use 1 inch standoffs. If mounted facing up at the bottom of the board (which cools the power regulator better), use ½ inch standoffs.

Power-On (and reset) Defaults

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

- **15M** – Select all four motors for all actions (such as defining motor current)
- **0**= – Define all motors to be at location 0
- **127E –** Set all output holding registers to '1', for glitch-free switchover to output mode if needed.
- **0F –** All programmable ports are configured as inputs.
- **80K** – Set the "Stop OK" rate to 80 steps/second
- **0N** – Step pulse widths are 6.67 microseconds
- **0O** – Set the step-and-direction signals to their default polarities. Note that the R1K and S1K jumpers can redefine this setting.
- **8000P** – Set the rate of changing the motor speed to 8000 steps/second/second
- **800R** – Set the target run rate for the motor to 800 steps/second
- **0T** – Enable all limit switch detection
- **0t** – Disable 'Instant Stop' from the I/O 5 input pin
- **1V** – Set <CR><LF> sent at start of new command, abort data send if new incoming command
- **0d** – Set DAC 0 to 0 volts
- **256d** – Set DAC 1 to 0 volts

USB Driver Installation Under Windows

Our USB-based boards use a USB driver chip for communications with your hosting computer. FTDI (http://www.ftdichip.com) provides drivers for operation under Windows[tm], Linux, and Mac/OS. 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 SD4DP 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 SD4D board is NOT on any sort of conductive surface (for example, metal, your hand, a carpet) when you do this, since you could damage the board (or your computer!) if any of the signals on the board get shorted.*** Note that you do NOT need to have the board connected to any external product (such as an actual motor driver) to install the drivers: just the SD4DP board and a USB A-B cable are needed, in addition to your computer with its USB 1.1 or 2.0 connection.
2. The SD4DP powers the USB portion of the board from the power available from the USB connection itself, and thus does not require board power for Windows to start the driver installation process.
3. This will cause Windows to bring up their "Found New Hardware" wizard, which will guide you through the installation process.
4. Place our installation CD into your CD drive.
5. If our setup application starts up, cancel out of it
6. Tell the wizard to "search for a suitable driver", and then tell it to "specify a location".
7. It will then ask for where to search: tell it to look in the "FtdiStepperBoard" directory on our support CD.
8. Then tell it to install the driver. If you are installing from the 'FtdiStepperBoard' version of the drivers, then Windows will complain that the drivers are not 'Windows Certified'. You may ignore the error; all that is different between the 'ftdi' certified installation and the 'FtdiStepperBoard' non-certified installation is that the installation script sets the communication defaults to our recommended values (below) and adjusts the list of recognized devices to include our products.
9. The installation may then go through the same process in order to install the virtual COM drivers (if you have never installed an FTDI USB-based product before). Use the same subdirectory and process to install those drivers as were used under step 7, above.

10. Once that process completes, the code will automatically add a new "COM" serial port that is "attached" to the board when it is plugged into the ***any*** USB port on your computer. *The system will automatically add a new COM port each time you attach a new board to any USB port on your computer or hub. It may also create a new COM port if you receive a repaired board back from us (if we have had to replace the USB driver chip).*

Initial testing of the board after driver installation – TestSerialPorts

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

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 (for our USB based boards) for TestSerialPorts to be able to locate the board).

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

Adjusting Default COM port properties for best operation

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

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

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

TTL Signals

The TTL input system normally provides for 15 input signals. 7 of the input signals may be individually redefined as outputs, if that is required for your application (see the 'F' command on page 28 for more information). TTL based control operates at the same time as serial control; therefore, any of the actions listed below may be requested at any time that the board is not in its special "direct computer control" mode of operation.

All external connections are done via labeled terminal block (or snap-in) connections and one USB serial port on the "bottom" of the board. Most of the input and control signals are on the one side, while all of the motor and power connections are on the other side, as are some generic TTL I/O lines.

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

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

Additionally, if you actually do drive signals at noticeable current levels, the SX/48 chip will get warm, and may get hot. You may find that you have to cool the board (see our section about fan cooling the board, on page 12) if the SX/48 seems to be running too hot.

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

TTL Input Voltage Levels: Treated as 5 volt CMOS

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

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

Input Limit Sensors, lines LW- to LZ+

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

The connections are:

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

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

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

As the most extreme example possible:

- if the motor is currently running at its maximum rate of 60,000 steps per second,
- and the allowed rate of change of speed is 1 step per second per second,
- and the stop rate was set to 1 step per second,
- then the total time to stop would be 60,000 seconds (a little under 16.7 hours -- groan!), with a distance of ½ v^2, or ½ (60,000)^2, or 1,800,000,000 steps.
- Note that this same amount of time would have been needed to get up to the 60,000 rate to begin with…

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

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

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

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

General TTL signals collection 2: IO0 to IO6

Lines IO0 through IO6 are generic TTL I/O lines, programmable by you to be either input or output. They are normally configured as inputs, operated via microswitch closures to ground. Through use of the 'F' command, these signals can be redefined as outputs, thus providing up to 7 TTL-level output signals under computer control on the right side of the board.

These signals only have the 10-30K internal pull-ups available (which are enabled when the signals are configured as inputs). They do not have the additional external 1K pull-ups which are available on the SLEW and LIM input lines.

Serial Operation

The USB based serial control of the system allows for full access to all internal features of the system. It normally operates the standard 'full usb' rates (the drivers ignore any baud rate setting which you might use). 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".

Serial Commands

The serial commands for the system are described in the following sections. The code is case-sensitive (i.e., "d" means something different from "D"). ***Please be aware that any time any new input character is received, any pending output (such as the standard "*" response to a prior command, or the more complex output from a report) is cancelled.*** Additionally, for commands which have automatic waits built into them (such as "G" and "A"), the commands can be aborted, if more actions are still pending.

Serial Command Quick Summary

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

General Commands

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

Motor Control Configuration

Motor Selection

Motor Motion Configuration

Motor Arc Drawing

A – Draw an Arc of the requested radius
B – Select Beginning arc angle
C – Define the arc Count of steps targeting the 'run rate'
c – Define the arc count of steps targeting the 'stop-at' rate
D – Define the arc Delta angle per step

Motor Motion Control

G or g – Go to position w, x, y, z on all of the motor(s)
h – instant queue and motion status report
I – Wait for motor 'Idle'
i – Wait for command queue space
Q – Stop all motors ("Quit") instantly
q – Pause all motors (slow-down then stop); use 'r' to restart from pause
r – Restart from pause/limit switch action
W,w – Set next W value (absolute or relative)
X,x – Set next X value (absolute or relative)
Y,y – Set next Y value (absolute or relative)
Z,z – Set next Z value (absolute or relative)
Binary set relative W, X, Y, or Z
Binary set absolute rate
= Assign current W, X, Y, Z values as current location

Serial command synchronization

Most serial commands immediately send back an '*' to indicate that they have completed their action. Some commands, however, may take an arbitrary amount of time to execute. The core motion code in the firmware maintains a 4 element queue of pending requests. The A, G, R, W, X, Y, Z, = and binary set data commands all are obligated to wait until there is queue space available before they can operate; therefore, they will not send back their particular acknowledgement of the action (the '*' for R, W, X, Y, Z and =, a complete queue status report for A and G) until the request can execute. This can make it a little difficult for application code to keep synchronized with what is going on, therefore the following rules have been added to standardize behaviours:

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

   s#*

   's' is the status after the command starts

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

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

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

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

   'S' – board is stopped from a TTL stop event (see the 't' command)

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

   '*' ALWAYS completes ANY response

3. The W, X, Y, Z, R, =, and binary set data (second byte) commands all are auto-wait with an 'accepted' response of '*'. If there is no queue space, they will wait until there is queue space before executing. Note that '=' requires the motors to be completely idle in order to execute. Please be aware of a known deadlock: if the control system is 'Paused' (see the next item) and there is no queue space left ("P0*" status), then these commands will wait 'forever' – i.e., they will never send an '*', and will have to be aborted (or restarted per items under '5', below).
4. If the system becomes paused and there is no queue space, then a full status report ("P0*") will be sent (formatted per item 2, above) (if you have any pending status report or '*'), and the pending status report (or '*') for the currently executing command will be abandoned. You will need to restart ('r') in order to allow the command to complete. You will have to issue an 'h', 'i' or 'I' to get a status report which tells you that it has completed.
5. For ALL commands that wait, the allowed nested commands are (all case sensitive):
    a. 'h' – immediate status report ('s#*')
    b. 'I' – queue status report ('s#*') request to be sent WHEN MOTORS ARE FULLY IDLE (or paused)
    c. 'i' - queue status report ('s#*') request to be sent WHEN THERE IS QUEUE SPACE (or paused/no queue space)
    d. 'q' – pause the motor (immediately returns '*')
    e. 'r' – restart from pause (immediately returns '*')
    f. '~' - abandon ANY pending status report or queued '*' (used to resynch communications)

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

Optimization available: Allow full response to always be sent

By default, all commands send a response as part of their operation (see above). However, if there is a new character waiting in the receive buffer for the board, all responses are skipped: this means that you cannot normally enhance communication speed by sending a series of commands and then looking at a series of responses.

The 'V' command (documented here) uses bit 1 to control whether thus suppression of command response is to occur. If the bit is 0 (the default), then the firmware operates as described. If the bit is 1 (for example, by using '2V' or '3V' as your configuration command), then the firmware will always send complete responses to all commands – no data is dropped.

This permits you to send a series of commands, and then look at the full list of responses. This mode is best used when queuing a new motion request: send all of the rate and coordinate data, and then the 'G' command to queue the request. Then monitor the result stream to look at the responses; wait for the final response (which will be the status report from the G) before continuing on.

*Please note that you **MUST** read the responses regularly*: the buffer used to report responses is only 128 characters long, and the processor will hang once that buffer becomes full!

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

Possible combinations:

-n: Value is treated as -n

n: Value is treated as +n

+n: Value is treated as +n

Examples:

-100X – Set next Motor X target location to be -100

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

A – Draw an Arc of the requested radius

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

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

- This command can take a very long time, since it operates by drawing the requested number of straight lines in a "circular" pattern. It sends a status report (formatted identically to that generated by the 'h' command) once the last vector of the arc has been queued. If any new character except 'h', 'I', 'i', 'r', 'q' or '~' is received by the controller while drawing an "arc", then the arc drawing will be stopped at the next vertex. The 'h', 'I' and 'i' characters (see Idle Wait) may be used to see if the arc is still going; 'h' immediately echoes back a status repprt identifying the main action of the controller, while 'I' and 'i' delay their reports until motion is complete or queue space is available (respectively). The commands 'q' and 'r' may be used to pause or restart the arc. The '~' character may be used as a "spacer" for communications timing – the 'A' and 'G' commands ignore it, aside from canceling any pending status report.

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

  Note that execution of the 'A' command will change the current values saved by the 'B' and 'C' commands, and will reset the current X and Y parameters to the last X,Y value requested as part of drawing the arc. Also, if the 'S' parameter is non-0, then the current target rate will be reset to the start/stop rate ('K' command) by the time the arc completes.

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

non-0, it will then set the target rate to the start/stop rate, and then do 'c' arc segments as it slows down towards the start/stop rate.

As a simple annotated example (the '*' is sent by the controller as its 'ready for next command' response, the 'G0*' is usually sent when the Arc command has finished queuing its vectors),

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

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

```
*250W   - Set W and
*300Z   - Z center point to the current location, with 'W' being logical 'X', 'Z' being
logical 'Y'
*0c     - Tell the system that there will be 0 steps to draw; we only go to the 'start'
location
*32B    - Begin "degroid angle" is 32 (which is 45 degrees)
*945A   - Radius of Arc is 945, and draw. This draws a line of length 945 at a 45
degree angle
G3*
```

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

B – Select Beginning Arc Angle

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

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

C – Define the arc Count of steps

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

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

c – Arc 'Slow-Down' count

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

D – Define the arc Delta angle per step

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

d – Set a DAC voltage value

This command is used to set one of the two on-board DACs (Digital-to-Analog Converter) to a given voltage. For each DAC, 0 maps into 0 volts, while 255 maps into 5 volts.

Bits	***Value***	***Use***
0-7	0-255	DAC value
8	+256	0 means select DAC 0, +256 means select DAC 1

For example, to set dac 0 to 2 volts, and DAC 1 to 3 volts, issue the commands:

102D

409D

Note that the second command is 256 (select DAC 1) + 153 (3 volts, from 255 * (3/5)).

At power on or reset, both DACs get reset to 0 volts.

E – Send Data to Programmable TTL I/O Ports

This command is used to send data to any ports on the SD4DP which have been defined as output ports through use of the 'F' command (next). Any data sent to a port which is currently configured as an input port will be retained as the value to use when (if) that port is changed to being an output port, thus providing for a defined state change sequence if reprogramming of a port is needed.

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

Bit	***Value***	***Signal***
0	+1	IO0
1	+2	IO1
2	+4	IO2
3	+8	IO3
4	+16	IO4
5	+32	IO5
6	+64	IO6

For example,

32F

would set IO5 high, and all of the rest low, and

65535F

would set all output lines high

F – Define I/O Port Directions

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

The encoding is therefore:

Bit	***Value***	***Signal: 1=output, 0=input***
0	+1	IO0
1	+2	IO1
2	+4	IO2
3	+8	IO3
4	+16	IO4
5	+32	IO5
6	+64	IO6

For example, to define IO5 as an output port, issue the command

32F

To define IO0 and IO5 as outputs, you sum the 'values' for IO0 and IO5 (1 and 32 from above, respectively), giving you the command:

33F

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

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

The software will:

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

For example,

```
*200W
*1000X
*-25687Y
*43156Z
*G
3*
```

Would:

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

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

H – Specify the motor backlash amount

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

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

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

h – instant queue and motion status report

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

s#*

where:

s is the status:

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

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

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

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

'S' – board is stopped from a TTL stop event (see the 't' command); no further motion will occur until the 'Q' command is issued

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

* reports completion of the status report.

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

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

I Wait for motor 'Idle'

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

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

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

i – Wait for command queue space

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

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

J – Set Selected I/O port bits to '1'

This command allows you to selectively set a subset of the bits on IO0-IO6 lines to high (1). Simply form the correct sum from the following table, to tell the code which set of output bits to set to 1 (high).

Bit	***Value***	***Signal***
0	+1	IO0
1	+2	IO1
2	+4	IO2
3	+8	IO3
4	+16	IO4
5	+32	IO5
6	+64	IO6

For example, to set IO1 and IO5 to '1' while leaving the rest unchanged, send the command :

34J

Note that this command does not affect the current I/O direction definitions: defining a bit to be '1' does not change an input bit to be an output bit. To define the port I/O directions, use the 'F' command.

K –Set the "Start/Stop oK" rate

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

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

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

For example,

100k

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

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

32051K

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

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

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

k –Set the current instantaneous motor rate

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

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

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

The latched events reported are as follows:

Bit	***Value***	***Description***
0	+1	W- limit reached during a W- step action
1	+2	W+ limit reached during a W+ step action
2	+4	X- limit reached during a X- step action
3	+8	X+ limit reached during a X+ step action
4	+16	Y- limit reached during a Y- step action
5	+32	Y+ limit reached during a Y+ step action
6	+64	Z- limit reached during a Z- step action
7	+128	Z+ limit reached during a Z+ step action
8	+256	System power-on or reset ("!") has occurred
9	+512	'G' or 'A' interrupted by a new character receipt
10	+1024	W motor has missed steps due to a rate/step pulse overrun
11	+2048	X motor has missed steps due to a rate/step pulse overrun
12	+4096	Y motor has missed steps due to a rate/step pulse overrun
13	+8192	Z motor has missed steps due to a rate/step pulse overrun
14	+16384	Pause is pending (at least, may also be paused)
15	+32768	System is paused

The "missed steps" bits are used to tell you that the system had to skip one or more steps due to a prior pulse from a prior step still being generated. Each pulse will have the width specified by the 'N' command (defaults to 10.4 microseconds), and there must be time for at least one 'off' cycle (5.2 microseconds) before the next pulse starts. If this is not true, then the appropriate "missed step" flag will be set. You will either need to select a lower top rate (the 'R' command), or a narrower pulse width (the 'N' command).

For example, after initial power on,

```
L
```

Would report

```
L,256
*
```

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

```
L,4
*
```

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

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

- The low 8 bits contain the raw IO port (bits IO0 through IO6 on the IO connector).
- The next 8 bits state which of those bits changed since the last "l" command.
- The next 8 bits state which bits went low since the last "l" command.
- The high 8 bits state which bits when high since the last "l" command.

The latched events reported are as follows:

Bit	***Value***	***Description***
0	+1	IO0
1	+2	IO1
2	+4	IO2
3	+8	IO3
4	+16	IO4
5	+32	IO5
6	+64	IO6
7	+128	always 0
8	+256	IO0 had an edge change
9	+ 512	IO1 had an edge change
10	+ 1024	IO2 had an edge change
11	+ 2048	IO3 had an edge change
12	+ 4096	IO4 had an edge change
13	+8192	IO5 had an edge change
14	+6384	IO6 had an edge change
15	+32768	0
16	+65536	IO0 had a low edge change
17	+131072	IO1 had a low edge change
18	+262144	IO2 had a low edge change
19	+524288	IO3 had a low edge change
20	+1048576	IO4 had a low edge change
21	+2097152	IO5 had a low edge change
22	+4194304	IO6 had a low edge change
23	+8388608	0
24	+16777216	IO0 had a high edge change
25	+33554432	IO1 had a high edge change
26	+67108864	IO2 had a high edge change
27	+134217728	IO3 had a high edge change
28	+268435456	IO4 had a high edge change
29	+536870912	IO5 had a high edge change
30	+1073741824	IO6 had a high edge change
31	+/-2147483848	0

For example, after initial power on, with all signals on the IO connector High,

```
l
```

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

```
l,2130739071
*
```

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

```
1,127
*
```

If IO0 then went low, it could then report

```
1,65918
*
```

M – Select multiple motors for following '?' commands

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

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

For example,

15M0?

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

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

N – Set the Pulse width

"N" is used to specify the width of the step pulse and the lead-in delay for the pulse, in 3.3333333 microsecond units. The low 8 bits of the command specify the pulse width, the next 8 bits specify the lead-in delay (firmware version 4.1 and later only). The legal pulse width values are from 1 to 255: if a value of 0 is specified, then 1 is used. The command is bit-encoded as:

Bits	***Value***	***Description***
0-7	1-255	Pulse width, in 3.33 uSecond units. A value of 0 gets reset to a value of 1 by the firmware.
8-15	0 to 255	Lead-in pulse delay, in 3.33 uSecond units. To use, multiply your lead-in pulse delay by 256, and add it to the pulse width request.

For example, to specify a pulse width of 3, with a lead-in delay of 1, you would form the value:

3 + (256 * 1) ➔ 259

and send that resulting sum.

For firmware 4.1 and later, the "lead-in pulse delay" was added to support step-and-direction boards which either require a minimum "idle" time on their step pulse, or which have a minimum setup time for stabilization of the direction signal before a new step pulse starts. Although the normal '0' lead-in is normally about 2-3 microseconds, it is only guaranteed to be at least 260 nanoseconds. Setting the lead-in delay field allows you to increase this by 3.3 microsecond units, to make certain that the pulse shape matches the needs of your motor driver.

The default power-on pulse width is 6.67 microseconds (2 counts), with a 0 count for the lead-time delay. You need to set this width to the smallest value which will successfully instruct your step-and-direction motor controller to actually step; the larger the value, the lower the top speed of the system (since you have to be able to have pulse edges).

The minimum cycle time is going to be the pulse width+lead-in delay+ 3.33 microseconds. You will have to adjust your rate so that it does not exceed the maximum number of complete cycles which can be generated per second based on the minimum cycle time.

For N values greater than 4, we can calculate the maximum rate as follows:

ND = N + D (N = pulse width, D = lead-in delay count)

CycleTime = (ND+1) * 3.33333333 (in microseconds)

CyclesPerSecond = 1,000,000 / CycleTime

For example,

ND	**Cycle Time**	**Max CyclesPerSecond**
4	16.7	60000
10	36.67	27273

You will need to adjust the above calculation if your motor driver board requires a longer delay between pulses.

Our firmware will detect an overrun condition if your requested rate is too large for the requested peak width value; see the 'L' command for the error report.

O – Set step and direction signal polarities and mode of operation

This command sets the polarities of the step and direction signals from the board to the external motor driver board. Note that the 'R1K' and 'S1K' hardware jumper options emulate use of this command.

By default, pulses are generated as 'LOW-HIGH-LOW'; that is to say, the output voltage is initially 0, then goes to 5 volts for the period specified by the 'N' pulse-width command, and finally returns to 0 to complete the pulse. This command allows you to reverse that behavior to be 'HIGH-LOW-HIGH' for any combination of the step output signals.

Similarly, the direction signals are configured by default to be 0 = negative, 1 = positive. This command allows any of those definitions to be reversed.

As of firmware version 1.2, bit 8 of this command is used to define how the 'step' and 'direction' outputs are actually used.

By default, a value of '0' operates those signals as defined by their labels – 'Step' contains the step pulses, while 'direction' defines the direction of travel for the motor during that pulse.

A value of '1' changes those signals to be "step-per-direction" outputs. In this mode, the 'Step' outputs are used for pulses requesting steps in the **positive** motor direction, while the "direction" outputs are used for pulses requesting steps in the **negative** motor direction.

The following table summarizes the bit-encoded parameter values for this command:

Bit	***Value***	***Description: 0 = default, 1 = invert signal***
0	+1	Z Step
1	+2	Z Direction
2	+4	Y Step
3	+8	Y Direction
4	+16	X Step
5	+32	X Direction
6	+64	W Step
7	+128	W Direction
8	+256	0 means operate in step-and-direction mode, 1 means operate in step-per-direction mode

For example, to make the Z motor step pulses be low-going (instead of the default of high going), while the Y direction is reversed, you could issue the command:

9o

When installed, The 'R1K' jumper option emulates the effect of issuing the command:

85o (i.e., flip all of the step signals)

Similarly, installation of the 'S1K' jumper emulations the effect of issuing the command:

170o

Finally, if both of the R1K and S1K jumpers are installed, the equivalent command emulation is:

255o

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 60,000 steps per second per second. If a value of 0 is specified, the code forces it to have a value of 8000. If a value above 60,000 (or less than 0) is specified, the code will accept it, but will ramp unreliably (i.e., do not do it!).

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

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

250p500r0X2000Yg

would cause the following actual ramp behaviors to occur:

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

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

Q – Stop motors immediately, abort queued motions

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

q – Pause motors

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

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

This defines the run-rate to be used for the faster motor, and it sets or clears the internal "trapezoidal mode" of motion profiling. The rate may be specified to be between 1 and 60,000 steps per second. If a value of 0 is specified, the code sets the rate to the default value, which is normally 800. If a value outside of the limits is specified, then it is accepted, but the code will not operate reliably. As with the slope ("P") rate, do not specify values outside of the 1-60,000 legal domain.

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

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

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

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

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

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

For example,

250R

Sets the stepping rate to 250 steps per second.

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

When the current rate mode is trapezoidal (positive rate), 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.

When the current rate mode is non-trapezoidal (negative rate), the new value is immediately set as the new target rate, thus allowing for dynamic multiple adjustments to the rate profile of your motion.

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

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

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

S – Send SPI Data

SendSPIData is supported by all **SD4DFifoNCStepper** firmware versions. It allows you to send 1 to 24 bits of SPI data through the IO5 and IO6 ports, with any of the other IO ports being selected as the chip select bit for SPI.

Wiring:

```
IO5:  SCLK to SPI device
IO6:  DATA in to SPI device
<your selected IO bit>: SYNC- (or CS-) to SPI device
```

The board automatically redefines IO5, IO6 and your selected CS line as being outputs, and then sends the SPI data as needed. Upon command completion, the three outputs are left high, and left as outputs.

Special value note: If the selected chip select port is 5, 6 or 7, then no CS action is done by the code. In this case, it is up to you to do your own CS via some other technique.

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

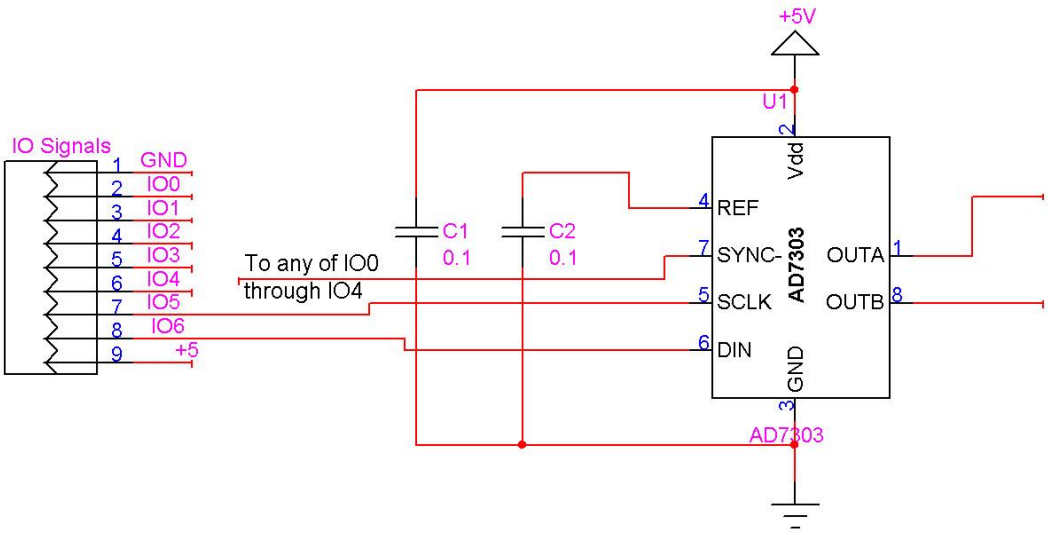
```
Bits    Use
0-4     cDataBits:  1 to 24

5-7     idCSPort:   Which IO line to use as the CS output
                     Value   Use
                       0      IO0
                       1      IO1
                       2      IO2
                       3      IO3
                       4      IO4
                       5, 6 or 7 No CS auto processing

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

Example of SPI to an AD7303

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



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

```
cDataBits: 16
idCSPort: 0 (connect SYNC- to IO0)
lData bits 0-7: DAC value (0-255 maps into 0 to 5 volts)
lData bits 8-15: microcode for AD7303:
                 3 means load dac A (OUTA)
                 7 means load dac B (OUTB)
```

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

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

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

You would therefore send the command:

471824S

in order to set OUTB to 1 volt.

Similarly, you would send the command:

261904S

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

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

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

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

T – limiT switch control

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

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

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

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

For example,

4T

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

65280T

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

t – Control TTL-based 'instant stop' input

The 't' command may be used to configure pin IO5 for use as an 'instant stop' input controller. When this feature is enabled, all motion will be instantly blocked when the voltage level on input pin IO5 is at the 'stop' level. The function provides for user control of whether the stop action is on a high or low TTL input.

Note that, in order to allow motion to resume after an instant stop, you will possibly have to disable this feature if your motion was blocked by some form of sensor (since there is no motion direction sensitivity provided, due to there being just one input sensor).

The command is bit encoded as:

Bit	Numeric Sum Value	Description
0	+1	Enable 'Instant Stop' if set to '1'
1	+2	Logic level which causes a stop event

This provides you with 3 usable values for 't':

- 0t -- Disable 'instant stop' (the default)
- 1t -- Enable 'instant stop', will stop on IO5 being low
- 3t -- Enable 'instant stop', will stop on IO5 being high

'1t' or '3t' will also automatically reprogram the IO5 pin as a TTL input.

Please note: IO5 has no pullups provided: you must provide a true TTL signal (0-0.4 volts low, 4.2 to 5 volts for high) in order for the input to be correctly processed.

Once a TTL stop event has occurred, the board's status response will go to 'S' (such as '3S*'). The only way to exit this mode is to send the 'Q' command -- you cannot resume ('r') from an instant-stop event.

U– Set Selected I/O port bits to '0'

This command allows you to selectively set a subset of the bits on IO0-IO6 lines to low (0). Simply form the correct sum from the following table, to tell the code which set of output bits to set to 0 (low).

Bit	*Value*	*Signal*
0	+1	IO0
1	+2	IO1
2	+4	IO2
3	+8	IO3
4	+16	IO4
5	+32	IO5
6	+64	IO6

For example, to set IO1 and IO6 to '0' while leaving the rest unchanged, send the command :

66J

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

V – Verbose mode command synchronization

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

It also controls whether most commands are 'aborted' by new pending input data. By default, if a long report is being generated (such as a '0?' output), it will be automatically abandoned if new input commands are seen. Similarly, if multiple requests are queued (such as setting output port values), the '*' responses will be dropped for all but the last one in the input buffer.

Through use of bit 1, this behavior can be changed to 'always send the complete report'. In this case, the complete command set will always be sent. PLEASE BE FOREWARNED! If you enable this state, and you then do not retrieve the transmitted data from the board, the system will eventually 'hang' while waiting for you to clear the pending 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	Always send pending data
2	+4	If set, report all data in signed hexadecimal instead of signed decimal (version 4.0 of the firmware and later!)

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, combined with always sending complete reports.

The Version 4.0 firmware enhancement of allowing generation of hexadecimal data reports changes both the values reported from being decimal to hexadecimal, and changes the separation character from being "," to being ";".

For example, a decimal report of current location could be:

```
M2,-1,-1000
```

If the above report is done when hex mode reporting is turned on, this would be reported as:

```
M2;-1;-3E8
```

This enhancement was added to support high-speed location reporting while motor motion is under way. Generating a decimal location report could cost the firmware a millisecond or so, which could cause a lag in starting the next segment of a multi-segment chained motion. Sending hex data does not experience this delay.

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

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

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

For example,

100W

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

100w

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

PLEASE NOTE A BEHAVIOR CHANGE: As of version 4.0 of the firmware, the following change was made:

As an intended side effect, the motor selected to receive the updated location also gets selected as if its equivalent 'M' command had been issued. This allows you to do the '-6?' report to confirm that the parameter value was set correctly, and the '-1?' report to see the current real motor location for the selected motor.

Binary Set W, X, Y, Z and Rate

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

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

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

Byte 1:

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

PLEASE NOTE A BEHAVIOR CHANGE: As of version 4.0 of the firmware, the following change was made:

As an intended side effect, the motor selected to receive the updated location also gets selected as if its equivalent ‘M’ command had been issued. This allows you to do the ‘-6?’ report to confirm that the parameter value was set correctly, and the ‘-1?’ report to see the current real motor location for the selected motor.

Byte 2:

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

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

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

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

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

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

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

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

' <so send the data…>
```

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

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

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

2000X

4000Y

=

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

! – RESET – all values cleared. Duplicates Power-On Conditions!

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

For example,

!

resets the system to its power on defaults.

The reset command also selects the following settings:

- **15M** – Select all four motors for all actions (such as defining motor current)
- **0**= – Define all motors to be at location 0
- **127E –** Set all output holding registers to '1', to provide glitch-free toggling between input and output modes
- **0F –** All programmable ports are configured as inputs.
- **80K** – Set the "Stop OK" rate to 80 steps/second
- **0N** – Step pulse widths are 6.67 microseconds
- **0O** – Set the step-and-direction signals to their default polarities. Note that the R1K and S1K jumpers can redefine this setting.
- **8000P** – Set the rate of changing the motor speed to 8000 steps/second/second
- **800R** – Set the target run rate for the motor to 800 steps/second
- **0T** – Enable all limit switch detection
- **1V** – Set <CR><LF> sent at start of new command, abort data transmission if there are new unprocessed incoming commands
- **0t** – Disable 'Instant Stop' from the I/O 5 input pin
- **0d** – Set DAC 0 to 0 volts
- **256d** – Set DAC 1 to 0 volts

? – Report status

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

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

1-255: Report memory location 1-255

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

- 5: Port A register – this contains serial I/O
- 6: Port B register – this contains the limit inputs
- 7: Port C register – this contains raw USB data
- 8: Port D register – this contains the step-and-direction signals
- 9: Port E register – this contains the IO0 to IO6 signals

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

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

- -1; Report current location
- -2; Report current speed
- -3; Report current slope
- -4; report target position
- -5; Report target speed
- -6; Report current 'request' scratch-pad value
- -7; report current 'pending' target location
- -8; Report step action (i.e., motor state)
- -9; Report step style
- -10; Report run rate
- -11; Report stop rate
- -12; Report current software version and copyright
- -13 to –15; internal diagnostics
- -16; Report standard pulse width (the 'N' command parameter)
- -17; Report the bits which need to be flipped for the step and direction signals (the 'O' command)
- -18; report the current 'backlash' setting
- -19; internal diagnostic
- -20; report maximum allowed step rate
- -other: reserved – may do extra internal diagnostic reports, or acts like 0 (report all specials)

All of the reports follow a common format, of:

1. If Verbose Mode is on, then a <carriage return><line feed> ("crlf") pair is sent.
2. The "M" followed by a digit corresponding to the motor being reported on is sent (i.e., 'M2' for X, or "M3" for Y).
3. A comma is sent . If the special "Hex mode" output mode has been selected, this will be a semicolon.
4. The report number is sent (such as –4, for target position).
5. Another comma (semicolon if hex mode) is sent.
6. The requested value is reported.
7. If this is a report for multiple motors, then a <crlf> is sent.
8. If this is a report for multiple motors, each remaining report is sent.
9. If Verbose Mode is on, then a <crlf> is sent
10. A "*" character is sent.

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

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

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

The special reports which are available are as follows:

0: Report items '-1' through '-11'

The "report all reportable items" mode reports the data as a comma (semicolon, if hex mode reporting has been enabled through the 'V' command) separated list of values, for reports –1 through –11. Just after power on, for example, the request of "0?" would generate the report:

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

Where:

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

For example,

```
6M0?
```

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

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

-1: Report current location

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

For example,

```
6M-1?
```

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

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

-2: Report current speed

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

For example,

```
6M-2?
```

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

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

-3: Report current slope

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

For example,

```
6M-3?
```

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

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

-4: Report target position

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

For example,

```
6M-4?
```

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

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

-5: Report target speed

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

For example,

```
6M-5?
```

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

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

-6: Report scratch pad value

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

For example,

```
6M-6?
```

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

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

-7: Report pending target location

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

For example,

```
6M-7?
```

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

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

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

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

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

For example,

```
6M-8?
```

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

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

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

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

This report currently always returns a '0' on the SD4D series of boards.

-10: Report run rate

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

For example,

```
6M-10?
```

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

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

-11: Report stop rate

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

For example,

```
6M-11?
```

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

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

-12: Report current software version and copyright

This reports the software version and copyright.

For example,

```
6M-12?
```

could report:

```
*
SD4DFifoNCStepper.src 1.9
Copyright 2006 by Peter Norberg Consulting, Inc. All Rights
Reserved
*
```

-13 through –15: Reserved

These are reserved for internal use by the firmware.

-16: Report step pulse width

This reports the current step pulse width (the 'N' parameter value). The current code scales this by about 4.35, and treats the result as the number of microseconds to assign as the width for a step pulse.

For example,

```
1M-16?
```

could report:

```
*
M1,-16,2
*
```

-17: Report step and direction 'flip' bits

This reports the current XOR pattern to apply to the step-and-direction signals before placing them into the output port. This is the result of the 'O' command.

For example,

```
1M-17?
```

could report:

```
*
M1,-16,0
*
```

-18: Report current backlash setting

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

For example,

```
1M-18?
```

could report:

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

-20: Report maximum allowed step rate

This reports the maximum supported stepping rate for this firmware.

For example,

```
1M-20?
```

could report:

```
*
M1,-20,60000
*
```

other – Ignore, except as "complete value here"

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

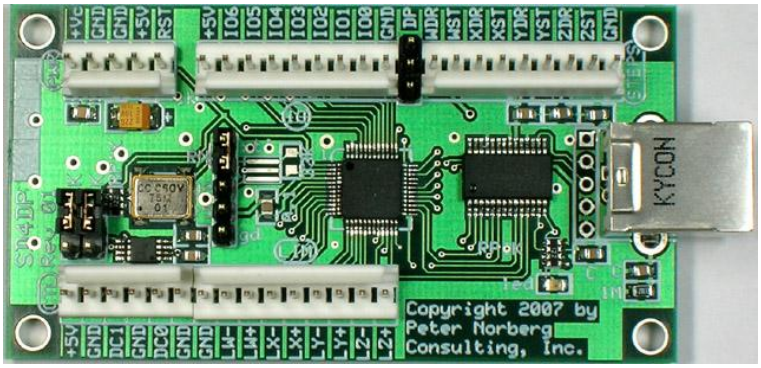
123 456G

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

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

Board Connections

An example of the SD4DP board is shown below.



Board Size

The board, oriented as shown on this page, is 3 inches wide by 1.6 inches high.

Mounting Requirements

The SD4DP 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. Horizontally, their centers are 2.75 inches apart, and vertically they are 1.35 inches apart. Thus, when the board is positioned as shown above, their positions are:

(0.125, 0.125), (2.875, 0.125),
(0.125, 1.475), (2.875, 1.475)

Connector Signal Pinouts

There are seven connectors on each board, given that 2 connectors (CTL and LIM) abut up against each other.

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

- SX-Key debugger connector (5 pin SIP header in middle of board)
- CTL – Output Power and DAC voltages (+5, GND, DC1 (DAC 1), GND, DC0 (DAC 0) GND.
- LIM – TTL Limit Input (GND, LW- to LZ+)

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

- PWR – Power and reset connector (upper left)
- IO – Generic TTL I/O connector
- STEPS – Step and Direction output connector
- USB Serial connector (center right hand side)

SX-Key debugger connector

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

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

CTL – Output Power and DAC voltages

Name	Description
+5	Access to +5 from the board
GND	Ground reference for all signals
DC1	DAC 1 0 to 5 volt output
GND	Ground reference for DAC 1
DC0	DAC 0 0 to 5 volt output
GND	Ground reference for DAC 0

This connector gives access to the board 5 volt regulator output, as well as the two 0 to 5 volt DAC outputs.

LIM - TTL Limit Input

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

The LIM connector is used to warn the SX-48 that a limit is being reached in the given direction. The signals are designed to be shorted to GND to denote that a limit is present; they are also compatible with normal TTL outputs from any TTL compatible device and are internally pulled up to +5 with 1K resistors.

The "6?" register report may be used to read the current status of these lines, while the "L" command may be used to report on whether any of these lines have actually triggered a limit-switch-based stop event.

PWR - Power Connector

Name	***Description***
+Vc	6.5-15 volts DC, to be regulated by the board
GND	Ground for Vc
GND	Ground for +5V
+5V	Either regulated 5 volts provided by you (do NOT connect anything to +Vc), or 5 Volt output from the board (max 100 mA draw)
RST	RESET- for processor. Pull low to initiate a board reset

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

1. If you have a 6.5 to 15 volt regulated DC power supply, then you can connect that to the +Vc and nearest GND connection. The on-board voltage regulator will regulate this down to 5 volts for use by the board, and will also present the 5 volts at the +5V/GND power connectors for external use (please do not draw more than about 100 mA).

2. If you have a regulated 5 volt DC power source, then you may power the board by connecting that supply to the +5V and nearest GND connector. In this case, do NOT connect anything to the +Vc; otherwise, you will have our on-board regulator and your +5 power supply both attempting to generate the board 5 volts, ***which will probably cause failure of our board (and possibly your power supply!).*** *This type of failure is not covered by our warranty.*

In both methods of powering our board, it is critical that you use a correctly grounded power supply, and that our GND power signal is also connected to true earth ground. ***If you fail to do this, you can have an incorrectly floating power system, which can cause failure of products (including damage to your computer) and can be potentially damaging to you!***

The RST input signal provides you with a hardware method of restarting the controller. By momentarily grounding the RST line (or by driving it low with a TTL signal), the board will act as if a power-on request has been made, thus resetting all of its internal states to match the power-on conditions.

IO - TTL Generic input and output signals

Name	Report Value	Description
+5		Access to board 5 volt regulator
IO6	+64	Generic TTL I/O 6
IO5	+32	Generic TTL I/O 5
IO4	+16	Generic TTL I/O 4
IO3	+8	Generic TTL I/O 3
IO2	+4	Generic TTL I/O 2
IO1	+2	Generic TTL I/O 1
IO0	+1	Generic TTL I/O 0
GND		Logic signal ground

This connector gives access to the board 5 volt regulator and generic TTL I/O lines.

The remaining signals (IO6 through IO0) are available for generic use by your application. They are controlled through use of the "E", "F", "J" and "U" commands, and their current values may be read through use of the "9?" report. The above table shows the binary weighting for the individual signal lines when presented in a report.

STEPS - TTL Step And Direction Signals

Name	Description
WDR	W motor direction
WST	W motor steps
XDR	X motor direction
XST	X motor steps
YDR	Y motor direction
YST	Y motor steps
ZDR	Z motor direction
ZST	Z motor steps
GND	Logic signal ground

This connector gives access to the actual step-and-direction signals as generated by the board. The outputs are configured in terms of polarity (high or low true) through use of the R1K and S1K jumpers, as well as through use of the 'O' command.

By default, these outputs are configured to be compatible with most step-and-direction motor drivers. The 'direction' signals define the direction of motor motion each time that an associated 'step' pulse event occurs. Under firmware versions 1.2 and later, a software command ("O", or via a special order option), these signals may be redefined to be "step-per-direction" signals In this mode, the "direction" outputs are reset to be "minus step pulses", while the "steps" outputs are reset to be "plus step pulses".

The outputs are buffered through a ULN2803 driver, which gives you adequate current to operate the Gecko products without additional external buffering.

The step signals all default to operate as 'Float-Pull Low-Float'; that is to say, a 'pulse' is defined as a signal which starts as 'floating' (value defined by your connection), is pulled low to GND for an indicated number of microseconds (as defined by the 'N' command), and then returns to a 'floating' state. If the 'S1K' jumper is installed at the time that the board is reset or powered on, then this sequence is inverted; the signal starts as being grounded, 'floating' for an indicated number of microseconds, and then returning to ground.

The direction signals are stable for at least 2 microseconds before the leading edge of the associated step pulse.

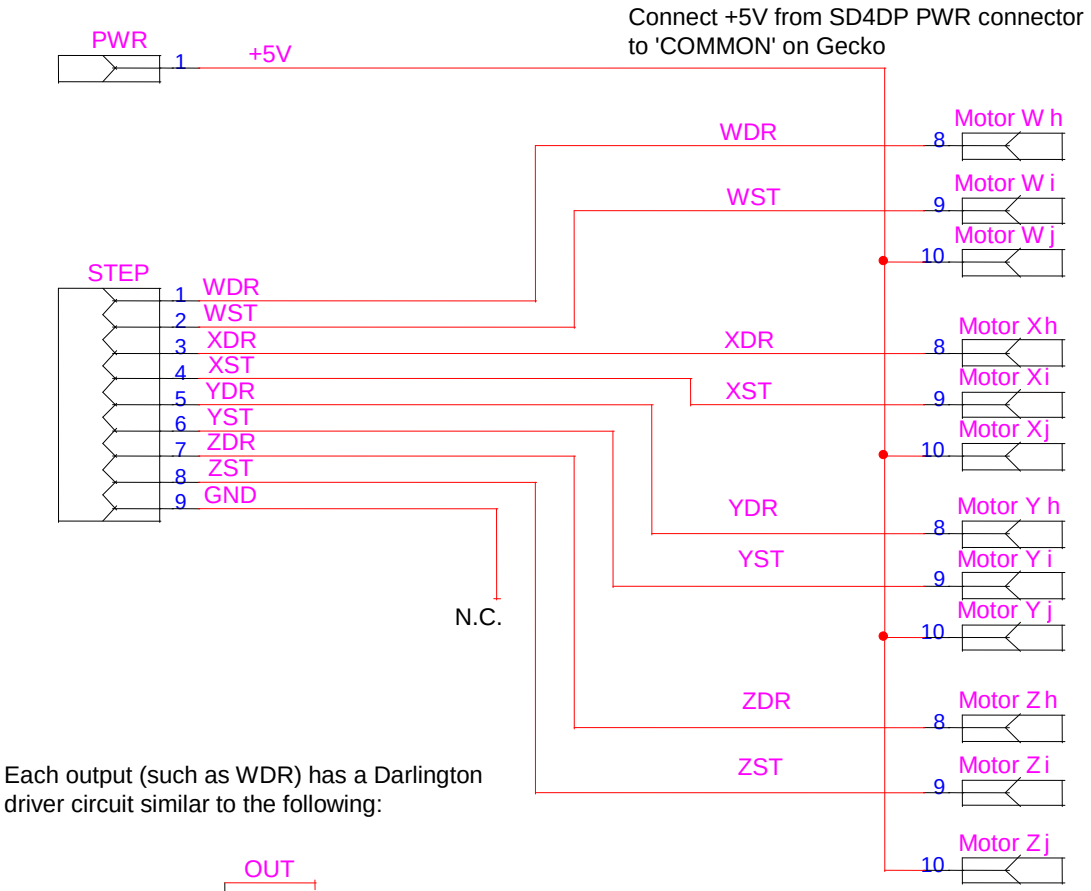
Many step-and-direction boards use optically isolated inputs to buffer the incoming signals. You will have to refer to the manufacturer's instructions to determine how to wire our ULN2803 driver to those boards. In most cases, our GND signal will be connected to their 'common', while our signal lines are connected to the appropriate step or direction inputs. In some cases (such as with Gecko G2O1 or G2O2 drivers), our +5V will need to be connected to their 'common' (due to the way that they have wired their optical isolators); our step and direction signals will still be connected to their appropriate signal inputs as needed.

You will then have to determine whether the polarity of our step pulses is correct: an incorrect choice can cause a missed step when a direction change is done, since the direction signal will be changed during the wrong state of the 'pulse'. Again, you will have to study the motor driver's manual to determine the correct settings; feel free to call us with questions, but we will probably need access to a copy of your documentation in order to be able to give you definitive answers.

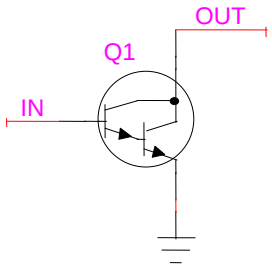
Connection Example 1: SD4DP to Gecko G201 or G2O2 Drivers

The following schematic shows how to connect the SD4DP controller to up to 4 Gecko G2O1 or G2O2 step-and-direction motor drivers.

Connection of the SD4DP controller to four Gecko G2O1 or G2O2 drives



Each output (such as WDR) has a Darlington driver circuit similar to the following:



This means that each output may be modeled as a switch closure to ground: The signal 'floats' when 'Off', and is grounded when 'On'

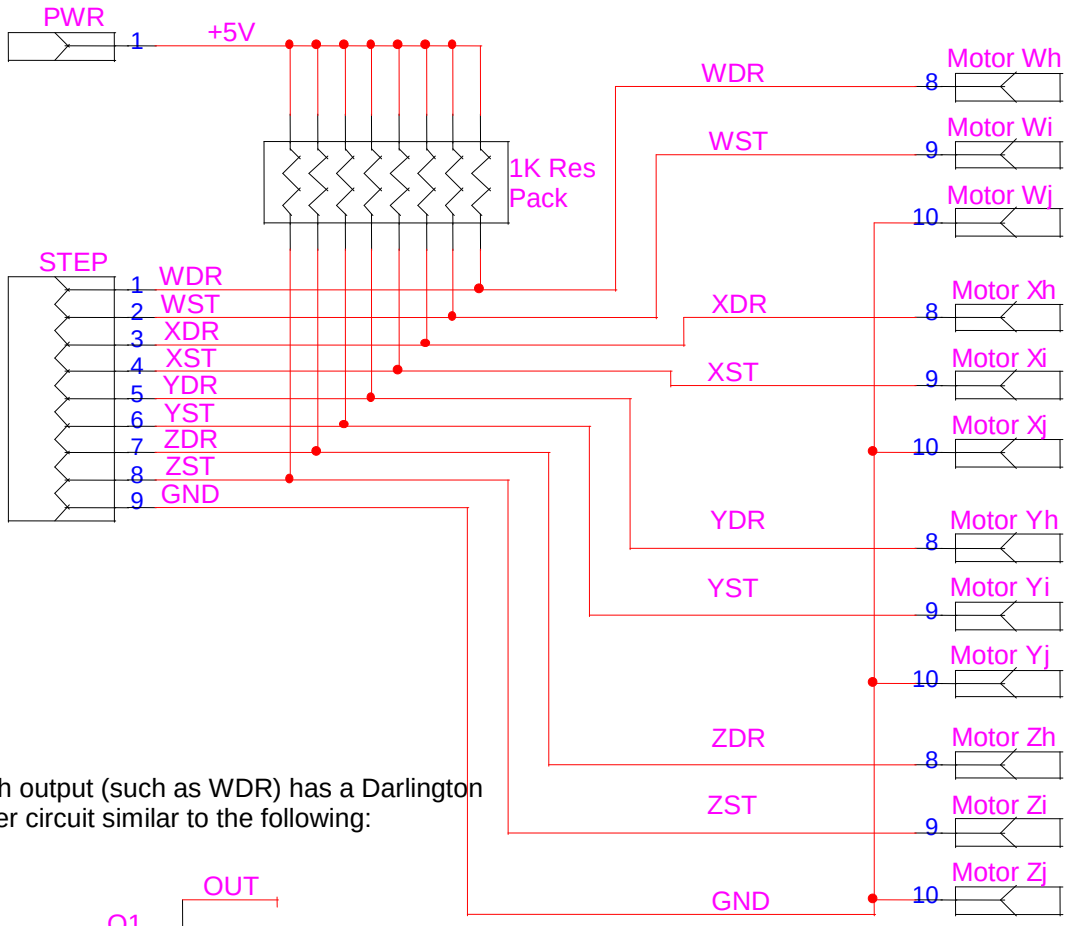
Connection Example 2: SD4DP to Gecko G203 Drivers

The following schematic shows how to connect the SD4DP controller to up to 4 Gecko G2O3 step-and-direction motor drivers.

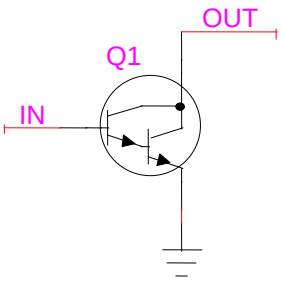
Please note that the SD4DP boards with serial numbers M209-0001 and above do NOT need the 1K resistor pull-up pack as is shown on this page – that pack is already mounted on the board.

Connection of the SD4DP controller to four Gecko G2O3 drives.

Unlike the G2O1 or G2O2, the G2O3 requires TTL input signals. We emulate this by using 1K pullups on our transistor outputs.



Each output (such as WDR) has a Darlington driver circuit similar to the following:



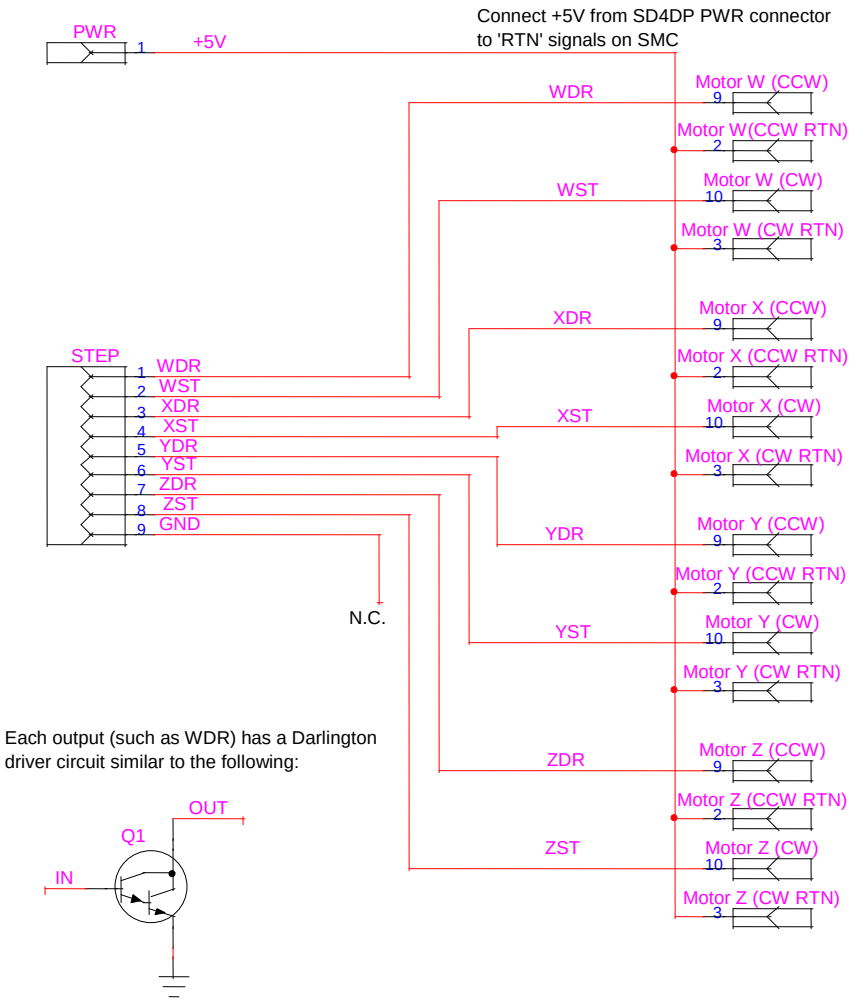
Connect GND from SD4DP SLEW connector to 'COMMON' on Gecko G2O3

This means that each output may be modeled as a switch closure to ground: The signal 'floats' when 'Off', and is grounded when 'On'

Connection Example 3: SD4DP to SMC LC6D Drivers

The following schematic shows how to connect the SD4DP controller to up to 4 SMC LC6D step-per-direction motor drivers. You will need to use the 'O' command to tell the board that the system is to be configured for "step-per-direction" mode of operation.

Connection of the SD4DP controller to four SMC LC6D drivers



Each output (such as WDR) has a Darlington driver circuit similar to the following:

This means that each output may be modeled as a switch closure to ground: The signal 'floats' when 'Off', and is grounded when 'On'