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 SD4DFifoStepper Firmware and the SD4DP series of Stepper Motor Controllers

By
Peter Norberg Consulting, Inc.
Matches SD4DFifoStepper Firmware Revision 1.5

Authored in the United States of America. Manual published June 13, 2011 11:08 AM

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 board artwork SD4DP. The firmware release described is SD4DFifoStepper version 1.3. The manual version shown on the front page normally has the same value as the associated SD4DFifoStepper 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
1.2	May 15, 2007	First manual release
1.3	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)
1.4	June 16, 2008	Corrected manual – summary section on R1K/S1K jumpers switched jumpers
	October 31, 2008	Corrected typo in manual
	November 20, 2010	Corrected typo in describing the 'L' command.
1.5	June 12, 2011	Extended pulse width command ("W") to allow setting of lead-in delay before pulse starts

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** step-and-direction motor controller from Peter Norberg Consulting, Inc., has the following general performance specifications:

	SD4DP
Maximum Logic supply voltage (Vc)	15V
Optional Logic supply voltage (+5V)	5V
Quiescent current	150 mA; 1 A if all drivers are active and driving a significant load
Board size	3.0" x 1.6"
ULN2803 Driver for Gecko drive compatibility	Yes

The SD4DFifoStepper firmware is designed to allow simultaneous independent control of up to 4 step-and-direction driver boards (such as those produced by Applied Motion or Gecko) using the SD4DP controller. board is operated via the USB serial interface, which provides full access to the controller's extreme range of stepping rates (1 to 50,000 steps per second), slope rates (1 to 50,000 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.

Short Feature Summary

- One through four step-and-direction stepper motors controllers may be independently controlled at one time.
- 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)
- Limit switches may optionally be used to automatically request motion stop of either motor in either direction.
- Rates of 1 to 50,000 steps per second are supported.
- Step rates are changed by linearly ramping the rates. The rate of change is independently programmed for each motor, and can be from 1 to 50,000 steps per second per second.
- All motor coordinates and rates are always expressed in logical full-step units. The controller has no knowledge of any microstepping which may be being performed by the actual external step-and-direction motor driver board(s).
- Motor coordinates are maintained as 32 bit signed values, and thus have a range of -2,147,483,647 through +2,147,483,647.
- Both absolute ("GoTo") and relative ("Slew") motion actions are fully supported.
- Complete control of the motors, including total monitoring of current conditions, is available through the USB connection.
- Runs off of a single user-provided 6.5 to 15 volt DC power supply or a single regulated 5 volt DC supply.
- The SD4DP includes a driver which has adequate current capabilities to drive the Gecko series of stepper controllers.

Firmware Configuration

The SD4DFifoStepper 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

The firmware can sense whether the 'Pot' resistor is present (actually, if the resistance present at the 'pot' connector exceeds the 10K maximum value). If it is missing, the firmware automatically selects a default slew rate of 800 steps/second. This default value can be ordered as any valid slew rate for the system (no command equivalent, but the command to set the current rate is 'R').

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 'D' command, which defines the port directions).

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 'C' 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 1.3 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 SD4DFifoStepper 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.

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):

- **15B** – Select all four motors for the following actions
- **0**= – Define all motors to be at location 0
- **0A** – Set DAC 0 to 0
- **256A** – Set DAC 1 to 0
- **127C –** Set all output holding registers to '1', to provide glitch-free toggling between input and output modes
- **0d –** All programmable ports are configured as inputs.
- **0h** – All motors have no backlash
- **80K** – Set the "Stop OK" rate to 80 **step**s/second
- **0O** – Set the step and direction polarities to standard
- **8000P** – Set the rate of changing the motor speed to 8000 **step**s/second/second
- **800R** – Set the target run rate for the motor to 800 **step**s/second
- **0T** – Enable all limit switch detection
- **1V** – Set <CR><LF> sent at start of new command, automatic cancel of data transmission on receipt of new input data
- **2W** – Step pulse width is 8 microseconds

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 'D' command on page 22 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 11) 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 "left" side of the board (limit switch and slew 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 LY- to LX+

Lines LY- through LX+ 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.

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 1K resistors.

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 50,000 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 50,000 seconds (a little under 13.9 hours -- groan!), with a distance of ½ v^2, or ½ (50,000)^2, or 1,250,000,000 steps.
- Note that this same amount of time would have been needed to get up to the 50,000 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!***

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.

Actual control of the stepper motors is performed independently for each motor. A "goto" mode is supported, as is a simple "go in a given direction". The code does support ramping of the stepping rate; however, it does NOT directly support changing the ramp rate, step rate, or "goto" target while a "goto" is under way. The behavior is either that the motor will *first* stop and *then* perform the new request, or that the new parameter value will be used on the *next* action. If button control is performed while a goto is underway, the goto gets changed to a direction slew, and the state of actions is reset.

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

1000G

would mean "go to location 1000"

0G?

would mean "go to location 0, and while that operation was pending, do a diagnostic summary of all current parameters".

Serial Commands

The serial commands for the system are described in the following sections. The code is case-insensitive (i.e., "s" means the same thing as "S"). ***Please be aware that any time any new input character is received, any pending output which has not actually started transmission (such as the standard "*" response to a prior command, or the more complex output from a report) is cancelled.*** This avoids loss of commands as they are being sent to the control board.

Serial Command Quick Summary

Most of the commands may be 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, SPI output

Motor Control Configuration

Motor Selection

Motor Motion Configuration

Motor Motion Control

All other characters – Ignore, except as "complete value here"

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

Possible combinations:

"-" alone – Set '-' seen, set no value yet: used on SLEW -

"+" alone – Clear '-' seen, set no value yet: used on SLEW+

-n: Value is treated as -n

n: Value is treated as +n

+n: Value is treated as +n

Examples:

-s – Start slew in '-' direction on the current motor

-10s – Slew back 10 steps on the current motor

A – 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:

102A

409A

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.

B – Select multiple motors

This command selects any combination of motors to be active, based on the value passed. 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,

15B0?

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

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

Note: the 'B' command is the only method available to control access to the 'W' and 'Z' motors. The 'X' and 'Y' commands provide alternative methods for accessing their respective motors. For example use,

1b

to select the 'W' motor, and

8b

to select the 'Z' motor.

C – Send Data to Programmable TTL I/O Ports

This command is used to send data to any ports on the SD4D which have been defined as output ports through use of the 'D' 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,

32C

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

127C

would set all output lines high.

D – Define I/O Port Directions

This command is used to define the I/O directions for all of the programmable I/O ports. It also will prevent any SLEW port which is defined as an output from causing a slew action to occur (the same effect as the 'u' command). Note that it does NOT re-enable slew access to any port redefined from being an output back to being an input; you need to use the 'u' command after redefining the input in order to enable TTL generation of slew requests.

This command is bit-encoded identically to the 'C' 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***
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

32D

To define IO0 and IO5 as outputs, you sum the 'values' for IO0 and IO5 (2 and 32 above, respectively), giving you the command:

34D

F – Send SPI Data

SendSPIData 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!

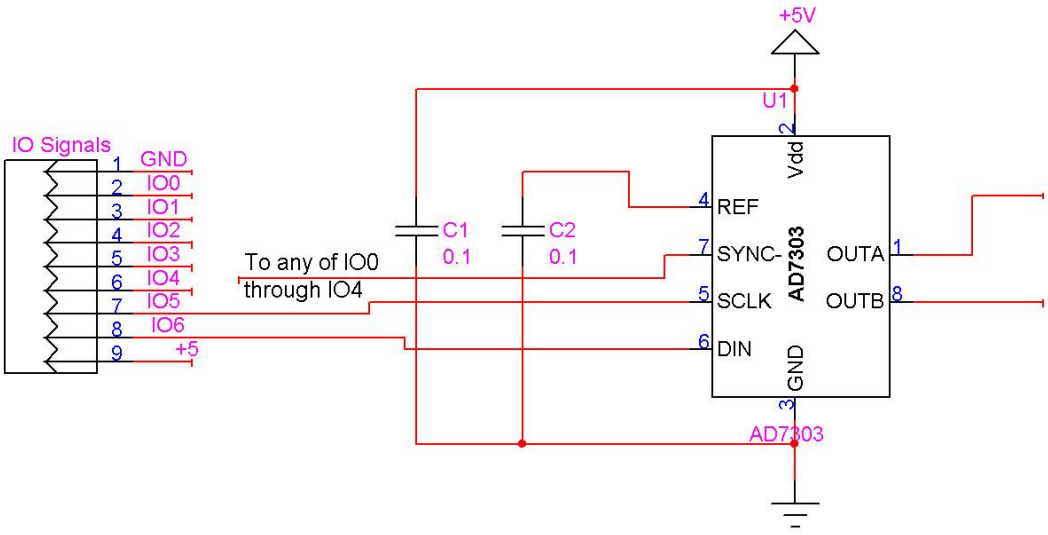
```
Bits    Use
0-4     cDataBits:  1 to 24

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:

471824F

in order to set OUTB to 1 volt.

Similarly, you would send the command:

261904F

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) ➔ 209680F
       OUTB: 458768 + (51 * 256) ➔ 471824F
```

G – Go to position x on the current motor(s)

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

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

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

For example,

X1000gy-25687g

Would:

1. Select the X motor for actions
2. Start a GOTO on motor X to location 1000
3. Select motor Y for actions
4. Start a GOTO on motor Y to location –25687

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

x-1?

Could report

M2,-1,350

*

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

H – Specify the motor backlash amount

'H' is used to specify (in steps) the motor backlash amount. The currently selected motor(s) (see the 'B' command) all get their backlash set to the specified value.

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.

I – Wait for motor 'Idle'

This allows your code to 'wait' for the currently selected motor(s) to (all) be idle. The code simply waits for either the selected motors to have completed their motion (see the **X**, **Y**, and **B** commands) or for the next serial character to be received, and then it transmits the '*' prompt (ready for next command). Note that, if the wait is stopped by

receipt of a new character, then the new character IS processed as part of a new command – it is NOT discarded.

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

Send	Receive
X	*
2000G	* *(note that the '*' is received as soon as the motion starts)*
I	* *(note that this '*' is not received until the motion completes)*

If you send a character before receipt of the final '*' (above), then system will discard transmitting the "*" response if it has not yet started the transmission. It will then process the new character. The best technique to avoid synchronization worries is to send two zero characters ('00'), wait for the second one to be completely sent, and then clear your input buffers. No further characters will be sent from the board until it sees the next command after this 'flushing' action (i.e., any pending data transmissions will be aborted).

J – Set Programmable TTL I/O Bits to 'High' levels

This commands 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 IO6 to '1' 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 'D' command.

K –Set the "Start/Stop oK" rate

This 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, 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 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.

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 W- step action
7	+128	Z+ limit reached during a W+ step action
8	+256	System power-on or reset ("!") has occurred
9	+512	<reserved: always 0>
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

For example, after initial power on,

```
L
```

Would report

```
L,256
*
```

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

```
L,4
*
```

M – Mark location, or go to previously marked location

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

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

x=1 : Go to last "marked" location

N– Set Selected I/O port bits to '0

This commands allows you to selectively set a subset of the bits on IO0-IO5 and the slew lines to low (0). Simply form the correct sum from the following table, to tell the code which set of output bits to set to 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 :

66N

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 'D' command.

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 'OFF-ON-OFF'; that is to say, the ULN2803 output voltage is initially off (floating), then ON (which shorts to ground) for the period specified by the 'N' pulse-width command, and finally returns to off to complete the pulse. This command allows you to reverse that behavior to be 'ON-OFF-ON' 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 1.3, 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 50,000 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 50,000 (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 currently have motor X selected, and it is at location 0, then the sequence:

250p500r2000g

would cause the following actual ramp behaviors to occur:

1. The motor would start at its "stop oK" rate, such as 80 steps/second
2. It 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).

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

This defines the run-rate to be used for the currently selected motor. It may be specified to be between 0 and 50,000 steps per second. If a value of 0 is specified, the code connects the rate to the user-provided potentiometer connected to the 'NXT' input. 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 1-50,000 legal domain.

This defines the equivalent number of steps/second which are to be used to run the currently selected motor under the GoTo or Slew command. The internal motor position is updated at this rate, using a sampling interval of 50,000 update tests per second. The motor windings are then updated according to the stepping mode.

The power-on/reset default Rate is 0, which causes the rate to be derived from the potentiometer setting

For example,

X250RY1000R

Sets the X motor target stepping rate to 250 steps per second, and the Y motor target rate to 1000 steps per second.

If you are currently executing a targeted GoTo or Slew command which has a specific target location (i.e., "2000g" or "-300s"), the new rate will not take effect until the motion has completed. If you are executing a generic "Slew in a given direction" command ("+s" or "-s"), the new rate will take effect immediately, and the motor will change its rate to match the request using the current "P" (slope-rate) value.

S – start Slew.

The "S"lew command is used to cause the currently selected motor to go in the selected direction. If the current value is only "+" or "-" (i.e., just has a sign associated with it), then the motor will slew in the indicated direction on the selected motor(s). Otherwise, the motor(s) will go VALUE steps in the direction indicated by the sign of VALUE, after first stopping the motor (more accurately, will target current location + x, then act as goto).

For example,

+s

will cause the current motor to start slewing in the forward direction, while

-250s

will invoke the "relative seek" calculation mode of the firmware.

When doing a relative seek (i.e., "-250s"), the address calculations are normally based on the current TARGET location, not the current instantaneous location. The actual rules are as follows:

1. If the given motor is currently executing a GoTo or relative Seek command, then the new location is calculated as a delta from the old target. For example,

```
Current State:
    Our current location is 1000
    Our current target is 2000
    We are doing a GoTo action
Request:
    -500s
Calculation:
    Since we are doing a normal GoTo,
    the new target location will be "2000-500", or 1500
Result:
    Motor stops, then goes forward to location 1500
```

2. Otherwise, the current location is treated as the value to calculate from for the relative motion. For example,

```
Current State:
    Our current location is 1000
    We are executing a "+s" command (slew positive)
Request:
    -500s
Calculation:
    Since we are executing a Slew,
    the new target location will be "1000-500", or 500
Result:
    Motor stops, then goes backward to location 500
```

This was set up this way as being a reasonable compromise on the intent of the meaning of "relative". If you want to force the motion to be strictly relative to the current location, you issue the "z" (stop) command first. Once that has been issued, the motor is placed in a special state (stopping, no target), which permits relative slew to be from the current location.

For example, to go -500 steps from the current location, regardless of whether the current action is a slew or a targeted goto, issue the command:

z-500s

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".

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

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.

W – Set the Pulse width

"W" is used to specify the width of the step pulse and the lead-in delay for the pulse in 4 microsecond units. The low 8 bits of the command specify the pulse width, the next 8 bits specify the lead-in delay (firmware version 1.5 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 4 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 4 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 1.5 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 4 microsecond units, to make certain that the pulse shape matches the needs of your motor driver.

The default power-on pulse width is 8 microseconds (2 counts). 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+ 4 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) * 4 (in microseconds)

CyclesPerSecond = 1,000,000 / CycleTime

For example,

ND	**Cycle Time**	**CyclesPerSecond**
4	20	50000
10	44	22727

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.

X – Select motor X

This command selects X motor as the target for the following commands.

For example,

X100r

Would cause the step rate to be set to 100 for motor X.

This command is identical to the 'B' command:

2B

Y – Select motor Y

This command selects Y motor as the target for the following commands.

For example,

Y100r

Would cause the step rate to be set to 100 for motor Y.

This command is identical to the 'B' command:

4B

Z – Stop current motor.

'Z' causes the current motor(s) to be ramped to a complete stop, according to its current ramp rate and stepping rate. "Stopped" is defined as "having a step rate which is <= the stop oK rate". See the 'K' command for defining the "stop oK rate".

For example,

Xz

Would slow down, then stop motor X.

! – 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. 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:

- **15B** – Select all four motors for the following actions
- **0**= – Define all motors to be at location 0
- **0A** – DAC 0 is set to 0 volts
- **256A** – DAC 1 is set to 0 volts
- **127C –** Set all output holding registers to '1', to provide glitch-free toggling between input and output modes
- **0d –** All programmable ports are configured as inputs.
- **0h** – Set motor backlash to 0
- **80K** – Set the "Stop OK" rate to 80 steps/second
- **0O** – Set the step and direction polarities to standard
- **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
- **2W** – Step pulse width is 8 microseconds

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

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

X2000=Y4000=

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 VALUE register, and issues an automatic stop ('Z') request. Note that the motor is stopped AFTER the assignment is complete, so the actual "current position" of the motor will be different from this value, depending on how long it takes for the motor to stop.

X2000=g

Would define the current location of the X motor to be 2000, and then would actually go to that 2000 location. This combination could be used when the motor is actually slewing or executing a "goto", to force the "current" location to be set and selected.

? – Report status

The "Report Status" command ("?") can be used to extract detailed information about the status of 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 7 MAY CHANGE BETWEEN CODE VERSIONS***

- 5: Port A register – this contains serial I/O, and the IO4-IO7 ports
- 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 IO7 signals

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

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

- -1; Report current location
- -2; Report current speed
- -3; Report current slope
- -4; report target position
- -5; Report target speed
- -6; Report standard pulse width (the 'N' command parameter)
- -7; Report the bits which need to be flipped for the step and direction signals (the 'O' command)
- -8; Report step action (i.e., motor state)
- -9; Report diagnostic motor flags
- -10; Report run rate
- -11; Report stop rate
- -12; Report current software version and copyright
- -13; Report motor backlash
- other: Treat as 0 (report '-1' to '-11')

All of the reports follow a common format, of:

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

If all 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 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 windings state (report "-6")
- g is the value for the stop windings state (report "-7")
- h is the value for the step action (motor state) (report "-8")
- i is the value for the diagnostic motor flags (report "-9")
- j is the run rate (report "-10")
- k is the stop rate (report "-11")

For example,

```
6B0?
```

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

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

-1: Report current location

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

For example,

```
6B-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,

```
6B-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,

```
6B-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,

```
6B-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,

```
6B-5?
```

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

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

-6: 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,

```
1B-6?
```

could report:

```
*
M1,-6,2
*
```

-7: 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,

```
1B-7?
```

could report:

```
*
M1,-7,0
```

*****

-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,

```
6B-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 motor diagnostic status bits

This reports the current diagnostic motor status bits. Normally, this report is not needed unless requested by our customer support staff.

-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,

```
6B-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,

```
6B-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,

```
6B-12?
```

could report:

```
*
SD4DFifoStepper.src version: 1.1
Copyright 2007 by Peter Norberg Consulting, Inc. All Rights
Reserved
*
```

-13: Report current backlash setting

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

For example,

```
1B-13?
```

could report:

```
*
M1,-13,231
*
```

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,

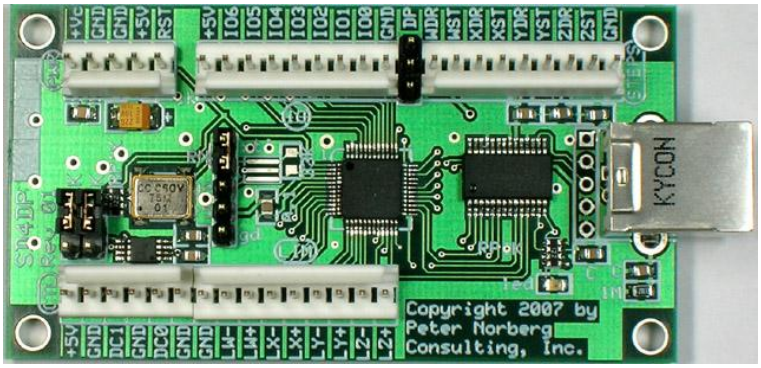
123 456G

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 "C", "D", "J" and "N" 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 1.3 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.

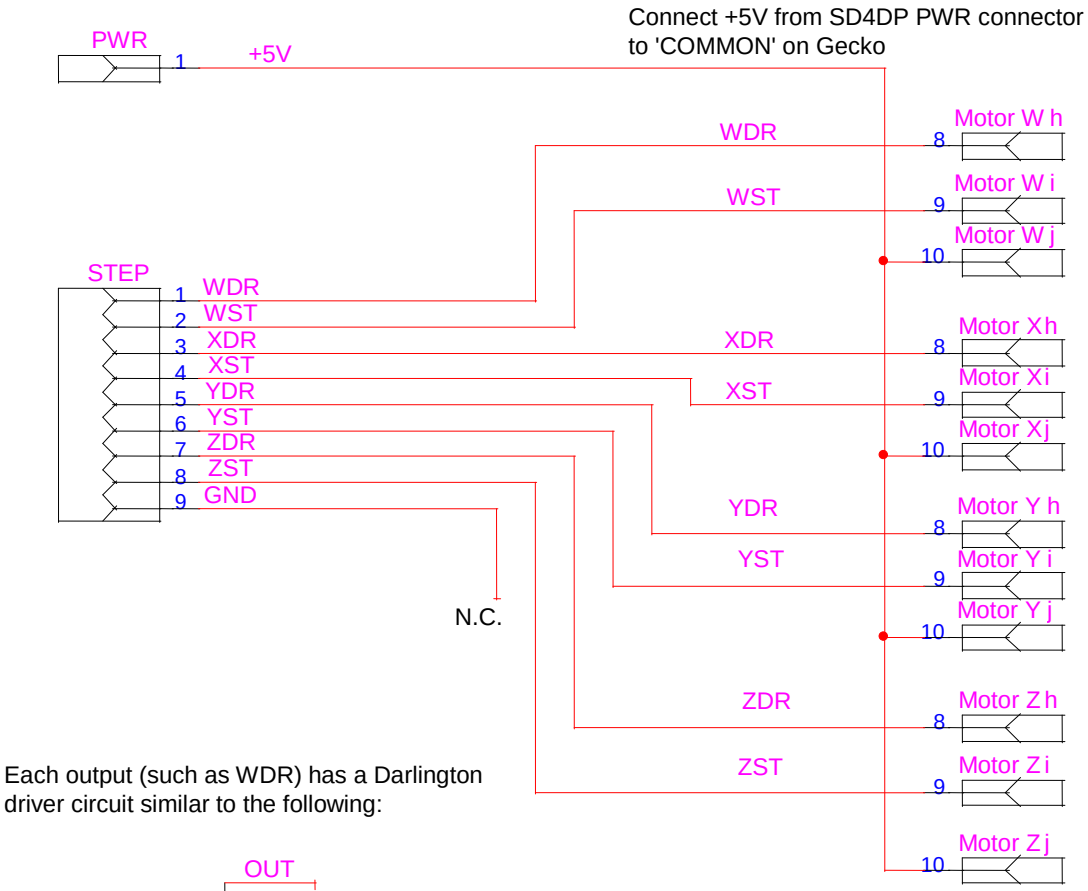
Many step-and-direction boards use optically isolated inputs to buffer the incoming signals. You will have to refer to the manufacturer's instructions to determine how to wire our ULN2803 driver to those boards. In most cases, our GND signal will be connected to their 'common', while our signal lines 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.

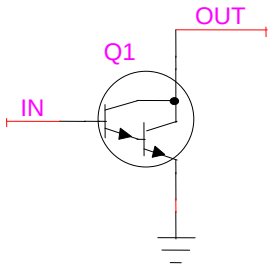
Connection Example 1: SD4DP to Gecko G201 or G2O2 Drivers

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:



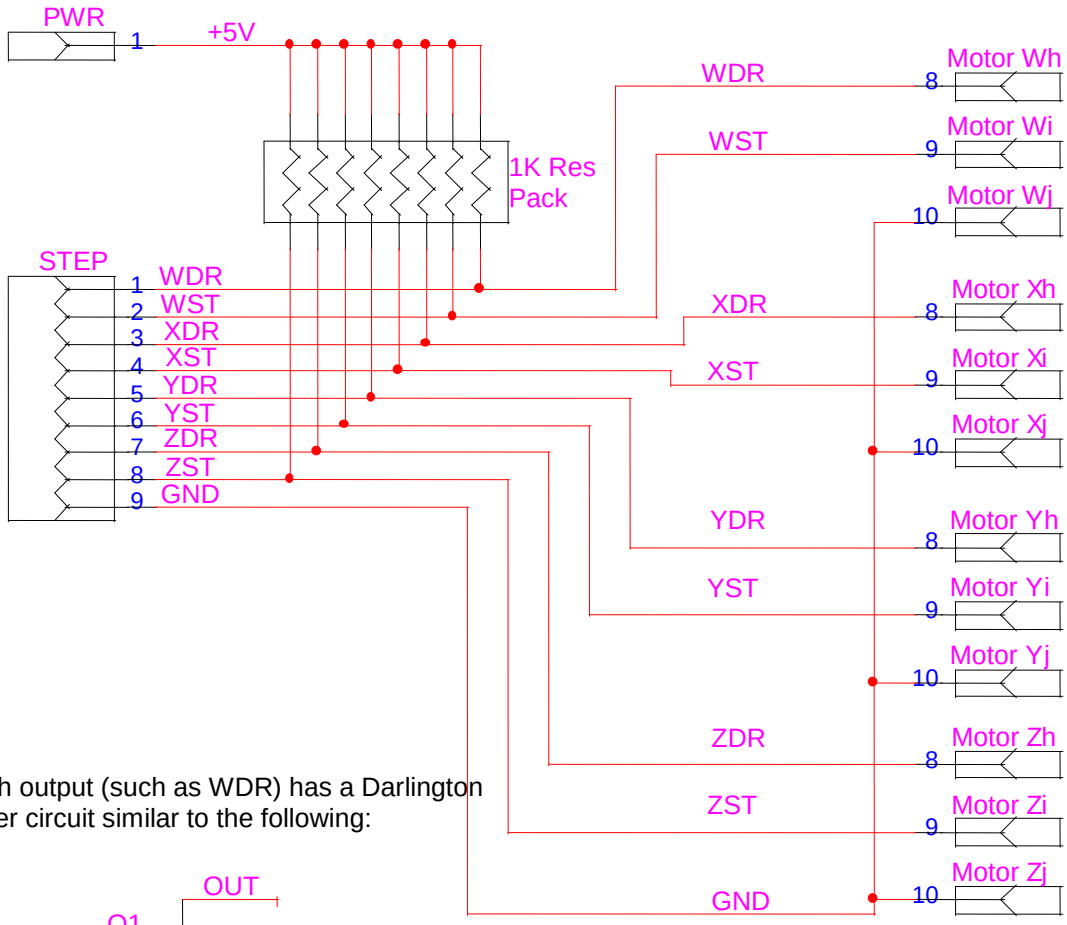
This means that each output may be modeled as a switch closure to ground: The signal 'floats' when 'Off', and is grounded when 'On'

Connection Example 2: SD4DP to Gecko G203 Drivers

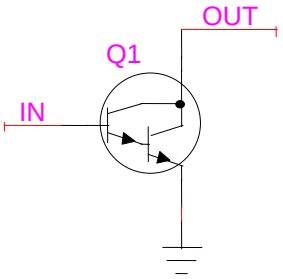
The following schematic shows how to connect the SD4DP controller to up to 4 Gecko G2O3 step-and-direction motor drivers.

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:



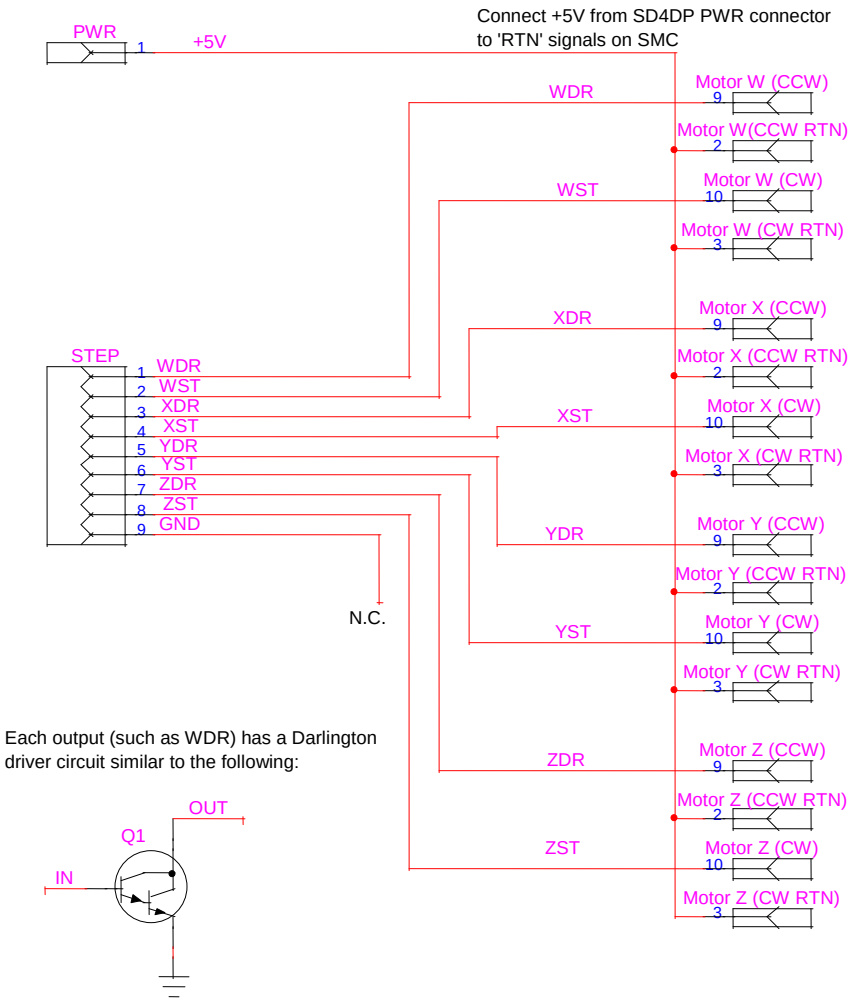
Connect GND from SD4DP SLEW connector to 'COMMON' on Gecko G2O3

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'