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 SD4DGenIO Firmware and the SD4D series of Stepper Motor Controllers

By
Peter Norberg Consulting, Inc.
Matches SD4DGenIO Firmware Revision 2.0

Authored in the United States of America. Manual published June 5, 2009 1:55 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 all of the SD4D series artworks, except for the SD4D. The firmware release described is SD4DGenIO version 2.0. The manual version shown on the front page normally has the same value as the associated SD4DGenIO 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
2.0	June 5, 2009	First manual release

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!**

Always unplug the board from the USB system and turn off any associated power supply when you are changing any connection to the board (such as unplugging a connector). The USB system always has some power present, and the SD4DEU model may be fully powered off of the USB**. It is unsafe to change connections to the board while it is powered!**

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

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

	SD4D01	SD4DG and SD4DEI	SD4DEU
Maximum Logic supply voltage (Vc)	15V	15V	15V
Optional Logic supply voltage (+5V)	5V	5V	5V
Quiescent current	150 mA; 300 mA if all drivers are active	150 mA; 1 A if all drivers are active and driving a significant load	150 mA; 1 A if all drivers are active and driving a significant load
Board size	3.0" x 1.6"	3.0" x 1.6"	3.0" x 1.6"
ULN2803 Driver for Gecko drive compatibility	No	Yes	Yes
Optional power via USB	No	No	Yes

The SD4DGenIO firmware is designed to allow simultaneous monitoring of up to 8 phase encoders, generation of 4 discrete pulse output signals, control of multiple SPI output devices, as well as generic TTL input and output (with 8 outputs being capable of driving low-actuating current relays).

The SD4DGenIO firmware builds upon many of the features of the SerRoute firmware available on our SS0705 series of boards. It removes the serial routing capabilities, electing instead to provide more comprehensive general I/O operations.

The system operates by your first setting up the parameters (such as the pulse count), and then executing a command (such as "G", for "Generate N pulses"). As a simple annotated example, the commands given could be as follows (the '*' character is sent by the controller as a "ready" prompt; the rest are commands sent):

```
*1B      - Select pulse counter 1
*1g      - pulse counter 1 uses high-current output 1
*3125W – Set both the 'high' and "low" pulse widths to be 25 milliseconds
*1O      - Make certain that we start with the relay 'open'
*1G      - Generate one 25 millsecond pulse
*65536T – Wait for the pulse to complete
*        - When this '*' appears, the pulse is done
```

The above sequence would generate one 25 millisecond wide pulse on high current output 1 (ZST), wait 25 milliseconds after the pulse completes (for a full cycle), and then report the '*' for 'operation done'.

Short Feature Summary

- Up to four separate pulse generators may be running concurrently, each operating any combination of the TTL or high-current output ports.
- Up to 8 phase-encoders may be simultaneously monitored
- 26 TTL I/O lines are available, most of which can be switched between being treated as inputs or outputs.
- When configured as outputs, instructions exist that allow you separately control the level (high or low) of each line.
- When configured as inputs, there exists an instruction which allows you to 'trigger' on a level change of any combination of the input signals (as well as on whether any of a set of pulse generators completes its sequence).
- 8 higher-current outputs (up to 200 mA per line) are available for driving LED's, relays, and optical isolators.
- 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, or off of USB power (SD4DEU only).
- Communication is via USB

Firmware Configuration: Jumpers and Special Orders

The SD4DGenIO firmware has a set of initial settings that are selected at power-on or reset that may be reconfigured at the time the product is ordered. All of these features may be reset through use of the appropriate serial command.

Default Serial Baud Rate

The USB drivers provided with the SD4D board emulate a serial (COM) port. The 'serial baud rate' is ignored by the drivers – communication always goes 'as fast as the board will accept it'

Jumper S1K: Default Initial I/O Port Directions

The board has a jumper ('S1K') which provides for two factory default selections on which ports are assigned as inputs and which are assigned as outputs after a board reset or power on sequence. By default, the two values for this option are configured as:

- S1K installed: '65295F' – Sets the "LIM" and "SLEW" connector signals to all inputs, the "IO" connector (IO0 through IO7) to all outputs.
- S1K removed: '65535F' – Sets the "LIM", "SLEW" and "IO" connector signals to all inputs.

At the time of ordering, you can specify the actual 'F' command to use for each of the above two jumper settings.

If you are going to be changing the 'LIM' 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.

Jumper R1K: Default TTL Output Values

The board has a jumper ('R1K') which provides for two factory default selections of the output signal levels which will appear for the drivers on the board after a board reset or power on sequence, assuming that the given driver is configured as being an output.

- R1K installed: '-256R' – All TTL outputs are high (on), all relay drivers (SD lines) are off.
- R1K removed: '0R' – All TTL outputs are low (off), all relay drivers (SD lines) are off.

At the time of ordering, you can specify the actual 'R' command to use for each of the above two jumper settings.

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 through 15).

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 your external devices 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 pulse generators for actions
- **0G** – Stop all four pulse generators
- **255E** – Select all 7 encoders for any future use of '='
- **0**= – Define all encoders to be at location 0
- **65535F –** Set I/O direction for the programmable I/O lines. Note: this actually is controlled by the 'S1K' jumper, as described on page 9.
- **0R –** All programmable outputs are set to OFF. Note: this actually is controlled by the 'R1K' jumper, as described on page 9.
- **1r** – Set RDY line HIGH
- **1V** – Set <CR><LF> sent at start of new command, no transmission delay time

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 SD4D 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 SD4D board and a USB A-B cable are needed, in addition to your computer with its USB 1.1 or 2.0 connection.
2. This will cause Windows to bring up their "Found New Hardware" wizard, which will guide you through the installation process.
3. Place our installation CD into your CD drive.
4. If our setup application starts up, cancel out of it
5. Tell the wizard to "search for a suitable driver", and then tell it to "specify a location".
6. It will then ask for where to search: tell it to look in the "FtdiStepperBoard" directory on our support CD.
7. Then tell it to install the driver. 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.
8. The installation may then go through the same process in order to install the virtual COM drivers (if you have never installed an FTDI USB-based product before). Use the same subdirectory and process to install those drivers as were used under step 7, above.
9. Once that process completes, the code will automatically add a new "COM" serial port 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.

The test assumes that the board is correctly configured to 'talk' to the com port: for the SD4D, the board must be correctly connected to the computer, and it must be powered on.

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 25 input signals and 9 output signals. The input signals may be redefined as outputs, if that is required for your application (see the 'F' 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.

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

TTL Input Voltage Levels: Schmitt-Triggered or CMOS

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

Note also that all of the TTL inputs are always internally tied to +5 via a very weak resistor (of the order of 5K- to 40K-Ohms). 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.

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 10) 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.

Inputs with pullups, lines LIM LW- to LZ+, and SLEW W- through Z+

Lines LW- through LZ+ and W- through Z+ are normally configured by default as TTL/Encoder inputs. They have an additional 1K pullup resistor installed on each input, in order to allow use of simple switch-closures-to-ground as the input signals. If they are to be used as outputs (see the 'F' command), please special-order your board from the factory with this 1K pull-up removed! Otherwise you will probably be be generating too much of a load on the microprocessor.

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.

General TTL signals collection: IO0 to IO7

Lines IO0 through IO7 are generic TTL I/O lines, programmable by you to be either input or output. They are normally configured as inputs, designed to be operated via microswitch closures to ground. Through use of the 'F' command, these signals can be redefined as outputs, thus providing up to 8 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.

Fixed TTL signals collection: POT input, RDY output

The signals 'POT' and 'RDY' (available near the LIM connector) are fixed in terms of being input or output.

POT is strictly an input signal, with very strong filtering (it uses a 1 uF capacitor to +5, followed by a 470 ohm resistor before it makes it to the microprocessor board). It may be viewed as bit 20 of the response to the '0T' command, and as bit 2 of the '-5?' raw register read command.

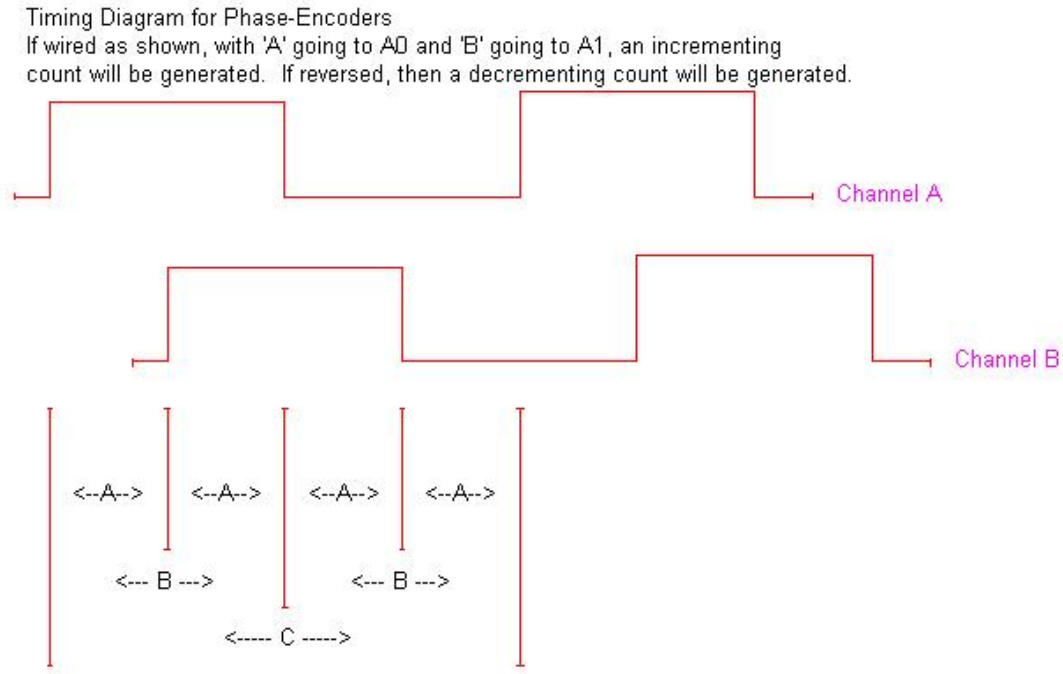
RDY is strictly a TTL output signal, capable of driving at most 5 mA of current. It may be controlled using the 'r' command.

Encoder Connections

Up to 8 phase-encoded positional encoders may be connected to most of the TTL-Signal lines, if the appropriate input mode is enabled (see page 22). The mappings are as follows: the 'Report' column describes which report command to issue in order to see the current encoder value.

Encoder	Signals	Report
0	LW-/LW+	-4?
1	LX-/LX+	-5?
2	LY-/LY+	-6?
3	LZ-/LZ+	-7?
4	IO0/IO1	-8?
5	IO2/IO3	-9?
6	IO4/IO5	-10?
7	IO6/IO7	-11?

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



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

For firmware version 1.8, 'x' is set to 8 microseconds.

Serial Operation

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

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,

1Bg

would mean "Select pulse counter 0, and assign it to driver output line ZST".

Serial Commands

The serial commands for the system are described in the following sections. The code is usually case-sensitive (i.e., "b" means something different from "B"). ***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 "T"), the commands can be aborted, if more actions are still pending.

On any command which is case insensitive (such as the "V"command), we strongly suggest that you just use the upper-case version of the command. Future extensions of this firmware may redefine the lower-case commands to some other use, if appropriate.

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

Pulse Generation Commands

Encoder Commands

Pulse counting

Generic SPI data transmission

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

xB – Select pulse generators to access for the subsequent commands

The 'B' command is used to select which of the four pulse generators are to respond to the following set of pulse-oriented commands. The 'x' parameter is encoded as:

Bit	***Value***	***Generator***
0	+1	0
1	+2	1
2	+4	2
3	+8	3

A total value of 0 is treated as 15; that is to say, '0B' selects all of the pulse generators.

xb – Set count to next pulse generator edge for current generator(s)

The 'b' command may be used to force a different start-time to the next edge of the current pulse (if the associated generator is currently running), or to the first edge of the first pulse (if the associated generator is no yet running).

Normally, the 'b' command is not used – When pulse generation is not on, the 'W' command forces the time to the first pulse to be effectively immediate (1 count), therefore you would only need to use this command if you are enabling multiple pulses, and you are attempting to delay one relative to another.

As with all of the pulse commands, the units are in terms of 8 microsecond clock cycles.

xC, c – Close (actuate) selected outputs (sets the output port bits to 1)

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

Bit	*Value*	*Signal*
0	+1	ZST
1	+2	ZDR
2	+4	YST
3	+8	YDR
4	+16	XST
5	+32	XDR
6	+64	WST
7	+128	WDR
8	+256	IO0
9	+512	IO1
10	+1024	IO2
11	+2048	IO3
12	+4096	IO4
13	+8192	IO5
14	+16384	IO6
15	+32768	IO7
16	+65536	LW-
17	+131072	LW+
18	+262144	LX-
19	+524288	LX+
20	+1048576	LY-
21	+2097152	LY+
22	+4194304	LZ-
23	+8388608	LZ+
24	+16777216	W-
25	+33554432	W+
26	+67108864	X-
27	+134217728	X+
28	+268435456	Y-
29	+536870912	Y+
30	+1073741824	Z-
31	-2147483648	Z+

For example, to set IO2 and ZDR to '1' while leaving the rest unchanged, send the command:

1026C

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.

xE, e – Select encoders which are to respond to the next '=' command

This command selects which encoders are have their values redefined by the next '=' command. If a bit related to the encoder is set, then it will be reset the next time an '=' is requested.

The encoding is therefore:

Bit	*Value*	*Signal: 0=output, 1=input/encoder*
0	+1	Encoder 0: LW- and LW+
1	+2	Encoder 1: LX- and LX+
2	+4	Encoder 2: LY- and LY+
3	+8	Encoder 3: LZ- and LZ+
4	+16	Encoder 4: IO0 and IO1
5	+32	Encoder 5: IO2 and IO3
6	+64	Encoder 6: IO4 and IO5
7	+128	Encoder 7: IO6 and IO7

For example, to reset the value reported by encoder 2 to be "123", issue the sequence:

4E

123=

xF, f – Define I/O Port Directions, Enable Encoders

This command is used to define the I/O directions for all of the programmable I/O ports. All ports are enabled based on their encoder assignment, with the data bit-encoded identically to that of the 'E' command. Any bit with a value of '0' defines the associated IO pair as being an output port. Any bit with a value of '1' defines that port pair as being an input port (with Encoder input processing automatically enabled)

The encoding is therefore:

Bit	*Value*	*Signal: 0=output, 1=input/encoder*
0	+1	Encoder 0: LW- and LW+
1	+2	Encoder 1: LX- and LX+
2	+4	Encoder 2: LY- and LY+
3	+8	Encoder 3: LZ- and LZ+
4	+16	Encoder 4: IO0 and IO1
5	+32	Encoder 5: IO2 and IO3
6	+64	Encoder 6: IO4 and IO5
7	+128	Encoder 7: IO6 and IO7
8	+256	W-
9	+512	W+
10	+1024	X-
11	+2048	X+
12	+4096	Y-
13	+8192	Y+
14	+16384	Z-
15	+32768	Z+

For example, to define IO0 through IO6 as outputs, LW- through LZ+ as 4 encoder inputs, and W- through Z+ as outputs issue the command:

15F

If you are going to be changing the 'LIM' or 'SLEW' inputs (bits 0 through 3 and 9 to 15, above) 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.

xG – Generate x pulses on current generator(s)

'G' is used to start (or stop) pulse generation on the currently selected and programmed pulse lines (from the 'B', 'W' and 'g' commands).

The parameter to 'G' tells the code how many pulses to generate:

0: Stop pulses which are ongoing

>0: Generate x pulses (complete low-high-low cycles)

-1: Generate pulses forever (no counting is done)

(other value <0): Reserved: current firmware will treat as '-1', but this may change!

Pulses are generated as 'XOR' patterns written against the current output registers. That is to say, one 'pulse' consists of two (timed) XOR requests, as defined by the 'W', 'w' and 'g' commands. 'W' controls the time between the first and second pulse, while 'w' controls the time after the second pulse (before the next sequence starts again). If you start with a given line being low, then the 'W' command defines the time that the pulse is high, while the 'w' command defines the time that the pulse is low if that time is different from the high time.

For example, to generate one 25 millisecond pulse on the ZST signal, you could issue the following sequence:

1B	Select pulse generator 0
1g	Define that generator to be accessing only the ZST signal
3125W	Set both the high-and-low times to be 25 milliseconds
1G	Generate one pulse

Note that only the 'W' command was given: when pulses are not yet being generated (you have not yet issued a non-0 'G' command for a given counter), 'W' actually sets both the high and low times to your requested value (thus generating a square wave), and it sets the begin time to be on the next clock cycle. For example, a '3125W' is exactly equivalent to the sequence:

3125W

3125w

1b

xg – Specify which I/O register bits are toggled by the current pulse generator(s)

'g' is used to select which of the 32 possible I/O lines are associated with the currently selected pulse counters.

Pulses are generated as 'XOR' patterns written against the current output registers. That is to say, one 'pulse' consists of two (timed) XOR requests, as defined by the 'W', 'w' and 'g' commands. 'W' controls the time between the first and second pulse, while 'w' controls the time after the second pulse (before the next sequence starts again). If you start with a given line being low, then the 'W' command defines the time that the pulse is high, while the 'w' command defines the time that the pulse is low if that time is different from the high time.

Note that it is explicitly legal to have multiple I/O ports selected as being managed by one pulse generator. The effect is that all of the lines will be 'toggled' at nearly the same time (within a few nanoseconds of each other). All lines on a given connector are toggled at precisely the same times, while ones on differing connectors will be offset somewhat.

The pulse generation is done by doing two XOR commands: the first is considered to be the 'High' part of the pulse (whose time is controlled by the 'W' command), while the second is considered to be the 'Low' part of the pulse (controlled by 'W', then overridden with 'w').

The bit definitions are the same as those used by the relay commands (such as 'C'), and are as shown in the following table.

Bit	***Value***	***Signal***
0	+1	ZST
1	+2	ZDR
2	+4	YST
3	+8	YDR
4	+16	XST
5	+32	XDR
6	+64	WST
7	+128	WDR
8	+256	IO0
9	+512	IO1
10	+1024	IO2
11	+2048	IO3
12	+4096	IO4
13	+8192	IO5
14	+16384	IO6
15	+32768	IO7
16	+65536	LW-
17	+131072	LW+
18	+262144	LX-
19	+524288	LX+
20	+1048576	LY-
21	+2097152	LY+
22	+4194304	LZ-
23	+8388608	LZ+
24	+16777216	W-
25	+33554432	W+
26	+67108864	X-
27	+134217728	X+
28	+268435456	Y-
29	+536870912	Y+
30	+1073741824	Z-
31	-2147483648	Z+

Therefore, to connect pulses to the 'ZDR' output, you would issue the command:

```
2g
```

This command does NOT affect the I/O directions: telling the code to write pulses to an input line is ignored!

To define the port I/O directions, use the 'F' command. The signals must be outputs for pulses to actually be generated.

Please be forewarned! The code explicitly permits use of the same I/O line between multiple pulse generators. If you have multiple generators running at the same time, the resulting pattern will be rather complex (since each generator will do its own XOR of the output data as needed).

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

The "L"atch reports allow 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 matching "L" or "l" command will only report new events since the last matching command.

The are two types of latch reports: "L" reports encoder overrun and reset events, 'l' (lower-case "L") reports I/O register events.

L – Encoder overrun, reset latch

The Encoder Latched events reported are as follows:

Bit	***Value***	***Description***
0	+1	Encoder 0 overrun
1	+2	Encoder 1 overrun
2	+4	Encoder 2 overrun
3	+8	Encoder 3 overrun
4	+16	Encoder 4 overrun
5	+32	Encoder 5 overrun
6	+64	Encoder 6 overrun
7	+128	Encoder 7 overrun
8	+256	System power-on or reset ("!") has occurred

The "overrun" bits are used to tell you that the system could not correctly interpret an encoder state change do to the pulses arriving too quickly. Each pulse cannot be less than 8 microseconds high and 8 microseconds low; if this is not true, then the appropriate "overrun" flag will be set.

For example, after initial power on,

```
L
```

Would report

```
L,256
*
```

xl – Latched I/O register events

The "l" latched I/O port command reports any state changes since the last call to "l" (lower case "L") for the selected I/O port 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 selected raw IO port.
- 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 legal values of 'x' are 0, 1 and 2. 0 requests the LIM register, 1 requests the IO register, while 2 requests the SLEW register. The low 8 bits of the report are thus:

Bit	***Value***	***x=0 LIM***	***x=1 IO***	***x=2 SLEW***
0	+1	LW-	IO0	W-
1	+2	LW+	IO1	W+
2	+4	LX-	IO2	X-
3	+8	LX+	IO3	X+
4	+16	LY-	IO4	Y-
5	+32	LY+	IO5	Y+
6	+64	LZ-	IO6	Z-
7	+128	LZ+	IO7	Z+

The higher level bits in the report are used to report change levels:

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

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

```
1l
```

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

```
l,-16711681
*
```

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

```
1l,255
*
```

If IO0 then went low, it could then report

```
1l,66047
```

*

xO, o – Open (release) selected relays

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

Bit	*Value*	*Signal*
0	+1	ZST
1	+2	ZDR
2	+4	YST
3	+8	YDR
4	+16	XST
5	+32	XDR
6	+64	WST
7	+128	WDR
8	+256	IO0
9	+512	IO1
10	+1024	IO2
11	+2048	IO3
12	+4096	IO4
13	+8192	IO5
14	+16384	IO6
15	+32768	IO7
16	+65536	LW-
17	+131072	LW+
18	+262144	LX-
19	+524288	LX+
20	+1048576	LY-
21	+2097152	LY+
22	+4194304	LZ-
23	+8388608	LZ+
24	+16777216	W-
25	+33554432	W+
26	+67108864	X-
27	+134217728	X+
28	+268435456	Y-
29	+536870912	Y+
30	+1073741824	Z-
31	-2147483648	Z+

For example, to set IO2 and ZDR to '0' while leaving the rest unchanged, send the command:

1026O

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.

xP – If TTL input mode is enabled, count pulses or measure pulse widths on any of the LIM input lines

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

The code operates as follows:

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

If requesting a pulse count (that is to say, the 'x' parameter is >= 0), then the number returned will be about 2x the "frequency" of the pulses, since both the leading and trailing edges are counted. For example, if you have a 1 kHz input signal, and you request a "10p" to sample for 10 milliseconds, the value reported will be 20 counts.

For example,

10p

could report

P,20

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

0P

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

The minimum detectable pulse width is 8 microseconds.

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

"-1P" Report starts at High going edge

"-2P" Report starts at Low going edge

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

For example, the resulting report of:

P,25,37

would mean that the low part was 25*8 microseconds wide, while the high part was 37*8 microseconds wide. The widths are all going to be +/- 1, since the code only samples on 8 microsecond intervals, and is thus asynchronous with respect to the real pulses.

The resulting report of:

P, 65423,0

would mean that the low part was 65,423 * 8 microseconds wide, while the high part timed out.

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

P,0,0

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

xp – Select Pulse Source Input Line

You can select the TTL input line which is used to provide the pulses for pulse count and width measurements. By default, at power on, line LW- is selected; however, any one of the eight inputs may be selected as the source of data for the next 'P' command through use of the 'p command.

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

Value	Pulse Source
0	LW-
1	LW+
2	LX-
3	LX+
4	LY-
5	LY+
6	LZ-
7	LZ+

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

```
5p
-1P
```

xR – Set all TTL outputs closed (1) or open (0) as requested.

This sets all outputs values shown, thus opening and closing all relays. The relays are assigned one per bit, as encoded in the following table:

Bit	*Value*	*Signal*
0	+1	ZST
1	+2	ZDR
2	+4	YST
3	+8	YDR
4	+16	XST
5	+32	XDR
6	+64	WST
7	+128	WDR
8	+256	IO0
9	+512	IO1
10	+1024	IO2
11	+2048	IO3
12	+4096	IO4
13	+8192	IO5
14	+16384	IO6
15	+32768	IO7
16	+65536	LW-
17	+131072	LW+
18	+262144	LX-
19	+524288	LX+
20	+1048576	LY-
21	+2097152	LY+
22	+4194304	LZ-
23	+8388608	LZ+
24	+16777216	W-
25	+33554432	W+
26	+67108864	X-
27	+134217728	X+
28	+268435456	Y-
29	+536870912	Y+
30	+1073741824	Z-
31	-2147483648	Z+

For example

7R

will set outputs ZST, ZDR and YST to ‘closed’; and will open all other TTL outputs (they will be set to 0).

xr – Set all RDY outputs closed high (1) or low (0) as requested.

This sets the TTL ‘RDY’ output to the low bit of the requested value.

For example,

0r

will set the RDY signal low, while

1r

will set it high.

S, s – Send SPI Data

SendSPIData is supported by all **SD4DGenIO** firmware versions. It allows you to send 1 to 24 bits of SPI data through the IO7 and IO6 ports, with any of the other IO ports being selected as the chip select bit for SPI.

Wiring:

```
<your selected IO bit>: SYNC- (or CS-) to SPI device
IO6:  SCLK to SPI device
IO7:  DATA in to SPI device
```

The board automatically redefines IO7, 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 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!

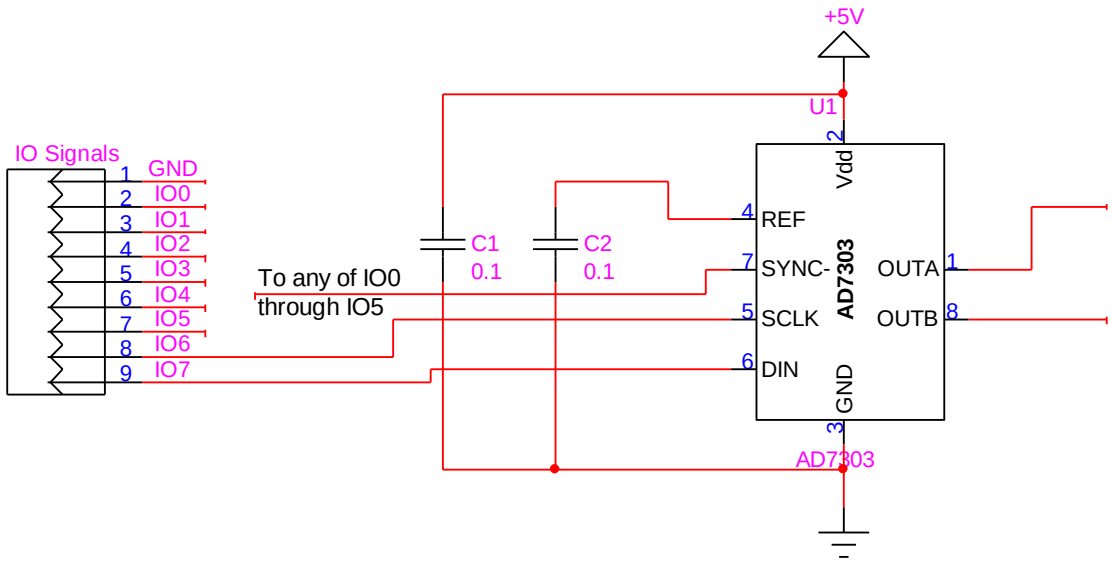
```
Bits    Use
0-4     cDataBits:  1 to 24

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     IO5
                        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, t – Wait for change of state on selected inputs (or pulse generation completion)

The 'T' command may be used to wait for a change in value of any combination of the 15 available TTL input lines, as well as for notification of completion of any of the 4 pulse generation sequences.

You issue the 'T' command with a numeric value formed from summing your request from the following table: the code will report which of those values changed when the change occurs (it will never report if no changes occur). Please note that you can abandon a pending 'T' command simply by issuing any other command (the 'T' will be forgotten).

Bit	***Value***	***Signal***
0	+1	LW-
1	+2	LW+
2	+4	LX-
3	+8	LX+
4	+16	LY-
5	+32	LY+
6	+64	LZ-
7	+128	LZ+
8	+256	IO0
9	+512	IO1
10	+1024	IO2
11	+2048	IO3
12	+4096	IO4
13	+8192	IO5
14	+16384	IO6
15	+32768	IO7
16	+65536	Pulse gen 0
17	+131072	Pulse gen 1
18	+262144	Pulse gen 2
19	+524288	Pulse gen 3
20	+1048576	POT
21	+2097152	0:reserved
22	+4194304	0:reserved
23	+8388608	0:reserved
24	+16777216	W-
25	+33554432	W+
26	+67108864	X-
27	+134217728	X+
28	+268435456	Y-
29	+536870912	Y+
30	+1073741824	Z-
31	-2147483648	Z+

For example, to wait on completion of either Pulse generator 0, or on any change to line LW-, issue the command:

65537T

The firmware will respond with the sum of the bits which changed, as:

T,1

(in this case, just LW- changed: it either went high or low relative to its state when the 'T' started).

A special value of '0T' may be used to get the current snapshot of all of the above registers (note that these values are also available as various queries using the '?' command)

0T

Could report:

T,6

which would claim that no pulse generators are busy, and only LW+ and LX- are high (all the other inputs are low).

xV – Verbose mode command synchronization

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

The verbose command is bit-encoded as follows:

Bit	SumValue	Use When Set
0	+1	Send <CR><LF> at start of processing a new command (enabled by default)
1	+2	<reserved for future use>
2	+4	Report data in hexadecimal instead of decimal
3	+8	Report data as unsigned instead of signed

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

1v

would enable transmission of the <CR><LF>sequence.

xW – Set pulse Width after first edge (high time)

W is used to set the pulse width after the first edge of any pulse generation event. If the given pulse generator is not yet running (i.e., a prior generation has completed or none has been started), then it also will set the pulse with after the second edge (acting like a 'w' command), as well as setting the time to the first edge (the 'b' command).

This command may be used while a pulse generation is occurring: it will simply redefine the 'time from the first to the second' edge on the next pulse.

The units are in terms of 8 microseconds each. This means that 1 millisecond may be specified by the value of 125: the maximum allowed count is 65535, which is 524.28 milliseconds.

If the pulse generator is not yet running, '125W' acts identically to:

125W	Set first-to-second edge time to 1 millisecond
125w	Set second-to-first edge time to 1 millisecond
1b	Set first edge occurs on the next clock interrupt

xw – Set pulse width after second edge (low time)

The 'w' command is used to set the time between the second edge of the pulse and the time when the pulse is complete (i.e., when the next pulse would start, if there are multiple pulses). If pulses are not being currently generated, the 'W' command will reset this value to match the first-second width (i.e., 'high' time of the pulse), thus generating a square wave. Therefore, to generate a non-square wave pulse, first specify the "W" value, and then the 'w' value.

For example, to request a pulse which is 32 microseconds high, 8 microseconds low (then the next pulse would start), issue the sequence:

4W	32 microsecond high
1w	8 microsecond low

As with the 'W' command, the 'w' command explicitly may be issued while a pulse generation sequence is occurring. The new 'w' value will take effect the next time that the code needs it, which is when it generates that second pulse edge.

x= – Assign current encoder value

If the appropriate mode is enabled via the 'F' command, then the requested lines are always monitored as if they were encoder inputs. This means that all transitions on any of those lines will result in updates to the seven 32 bit encoder counters. The '=' command allows those counters to be reset to user specified values. It works in conjunction with the 'e' command (on page 21), in that it assigns the requested 'x' value to the counters which have been selected through use of the 'e' command.

Please see the general value report command '?' (starting on page 40), for information on how to retrieve counter data.

For example,

```
255E
0=
```

will assign all counters a value of 0.

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

This command acts like a power-on reset. It ***IMMEDIATELY*** stops all pulse generators, and clears all values back to their power on defaults.

For example,

```
!
```

resets the system to its power on defaults.

The reset command also selects the following settings:

- **15M** – Select all four pulse generators for actions
- **0G** – Stop all four pulse generators
- **255E** – Select all 7 encoders
- **0**= – Define all encoders to be at location 0
- **65295F –** Set I/O direction for the programmable I/O lines. Note: this actually is controlled by the 'S1K' jumper, as described on page 9.
- **0R –** All programmable outputs are set to OFF. Note: this actually is controlled by the 'R1K' jumper, as described on page 9.
- **1V** – Set <CR><LF> sent at start of new command, no transmission delay time
- **1r** – Set RDY output high

? – 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 USB synchronization signals
- 6: Port B register – this contains the limit inputs (LIM)
- 7: Port C register – this contains SLEW inputs
- 8: Port D register – this contains the step-and-direction signals
- 9: Port E register – this contains the IO0 to IO7 signals

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

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

- -1; Report current location
- -2; Report current speed
- -3; Report current current board features (the "F" command)
- -4; Report encoder 0 counter value
- -5; Report encoder 1 counter value
- -6; Report encoder 2 counter value
- -7; Report encoder 3 counter value
- -8; Report encoder 4 counter value
- -9; Report encoder 5 counter value
- -10; Report encoder 6 counter value
- -11; Report encoder 7 counter value
- -12; Report current software version and copyright
- -13; Reserved as a diagnostic
- -14; Report currently running pulse generators
- -15; Report count of clocks per interrupt (75 Mhz clock)

All of the reports follow a common format, of:

1. If Verbose Mode is on, then a <carriage return><line feed> ("crlf") pair is sent.
2. A "S" character is sent, to signify that the report is from the SD4DGenIO board.
3. A comma is sent.
4. The report number is sent (such as –4, for encoder 0 value).
5. Another comma is sent.
6. The requested value is reported.
7. If Verbose Mode is on, then a <crlf> is sent
8. An "*" character is sent.

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

The special reports which are available are as follows:

0: Report all reportable items

The "report all reportable items" mode reports the data as a comma separated list of values, for reports –1 through –11 and -14 Just after power on, for example, the request of "0?" would generate the report:

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

Where:

- S identifies the report as SerRoute generated
- 0 is the report number; 0 is the 'all' report
- a is the value for the current relay settings (report "-1")
- b is the value for the current TTL input port data (report "-2")
- c is the value for the current feature selections (report "-3")
- d is the value for encoder 0 (on LW-/LW+, report "-4")
- e is the value for encoder 1 (on LX-/LX+, report "-5")
- f is the value for encoder 2 (on LY-/LY+, report "-6")
- g is the value for encoder 3 (on LZ-/LZ+, report "-7")
- h is the value for encoder 4 (on IO0/IO1, report "-8")
- i is the value for encoder 5 (on IO2/IO3, report "-9")
- j is the value for encoder 6 (on IO4/IO5, report "-10")
- k is the value for encoder 6 (on IO4/IO5, report "-11")
- l lists the current pulse generators (report "-14")

For example,

```
0?
```

Would report all reportable values for SD4DGenIO. You could receive:

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

which would mean:

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

-1: Report current relay settings

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

For example,

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

-2: Report current TTL input port data

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

For example,

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

```
*
```

-3: Report current feature selections

This reports the current "F"eature selections.

For example,

```
-3?
S,-3,15
*
```

-4: Report encoder 0 value

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

For example,

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

-5: Report encoder 1 value

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

For example,

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

-6: Report encoder 2 value

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

For example,

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

-7: Report encoder 3 value

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

For example,

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

-8: Report encoder 4 value

This reports the current value for encoder 4 (on IO0/IO1).

For example,

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

-9: Report encoder 5 value

This reports the current value for encoder 5 (on IO2/IO3).

For example,

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

-10: Report encoder 6 value

This reports the current value for encoder 6 (on IO4/IO5).

For example,

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

-11: Report encoder 7 value

This reports the current value for encoder 6 (on IO6/IO7).

For example,

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

-12: Report current software version and copyright

This reports the software version and copyright.

For example,

```
-12?
```

could report:

```
SD4DGenIO.src 1.8
Copyright 2007 by Peter Norberg Consulting, Inc. All Rights
Reserved
*
```

-13: Reserved – internal diagnostic report

This report is used internally as a diagnostic, and will change as needed.

-14: Report currently running pulse generators

This reports the encoded summary of current pulse generation events (same encoding as the 'B' command).

For example,

```
-14?
S,-14,1
*
```

-15: Report count of clocks per interrupt

This reports the count of clocks per interrupt cycle, based on a 75 MHz clock. This tells you the maximum frequency of pulse edges which can be detected or generated.

For example,

```
-15?
S,-15,600
*
```

other – Ignore, except as "complete value here"

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

12 5r

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

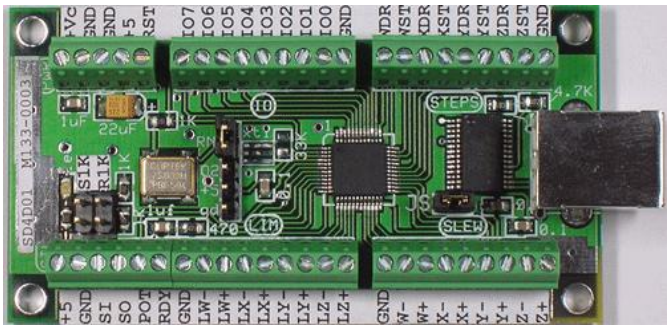
If verbose mode is enabled, then upon starting processing of any command (including the 'ignored' commands), the board sends the <carriage return><line feed> pair.

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

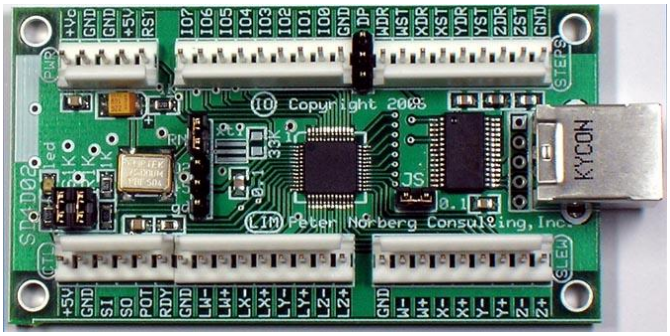
Board Connections

Board Connections

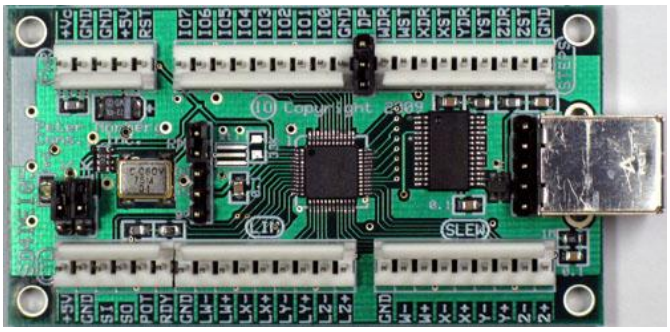
Examples of the SD4D series of boards are shown below.



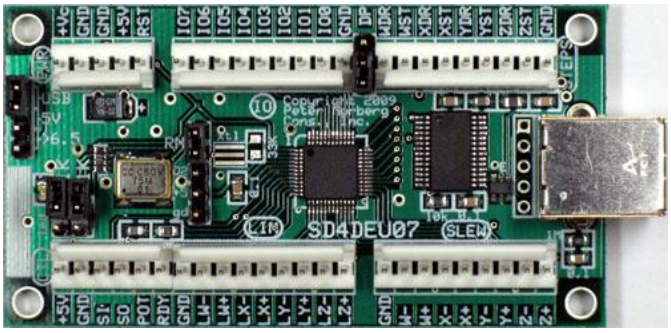
Above is the SD4D with screw terminal connectors



Above is the SD4DG with Amp MTA-100 connectors



Above is the SD4DEI with Amp MTA-100 connectors



Above is the SD4DEU with Amp MTA-100 connectors

Board Size

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

Mounting Requirements

The SD4D 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 eight 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 – Board status and TTL Serial (+5, GND, SI (Serial Input), SO (Serial Output), POT, RDY).
- LIM – TTL Limit Input (GND, LW- to LZ+)
- SLEW – TTL Motor Direction Slew Control (GND, W- to Z+)

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 - Board status and TTL Serial

Name	Description
+5	Access to +5 from the board
GND	Ground reference for all signals
SI	INPUT: Raw SX-48 Serial Input (TTL level)
SO	OUTPUT: Raw SX-48 Serial Output (TTL Level)
POT	Connect to potentiometer to control step rate
RDY	Ready/busy output

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

POT is unused when the SD4DNCRouter firmware is installed. In other firmwares, it can be connected to a user-provided potentiometer, to allow manual control of the motor stepping rate.

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

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

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

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.

SLEW - TTL Motor Direction Slew Control

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

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

W- through Z+ are inputs, normally used to control manual slew requests in firmwares which support that type of feature. In the SC4DNCRouter firmware, these are available as generic TTL inputs and outputs.

Normally, there is a 1K resistor pack installed as pull-ups for these signals. As an option at the time of the order, this resistor pack may be left off, if these signals are going to be used as outputs instead of inputs.

Please see the "E", "F", "J" and "U" commands for details of configuring these signals. Note also that the "7?" report may be used to read this set of inputs.

PWR - Power Connector for the SD4D, SD4DG, SD4DEI

This description is for the SD4D, SD4DG and SD4DEI boards. Please see the next page for the description of this connector on the SD4DEU board (which includes the option of being powered off of the USB system).

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.

PWR - Power Connector for the SD4DEU

This description is for the SD4DEU board. Please see the prior page for the description of this connector on the SD4D, SD4DG and SD4DEI boards.

Name	***Description***
+Vc	6.5-15 volts DC, to be regulated by the board
GND	Ground for Vc
GND	Ground for +5V
+5V	5 Volt output from the board (max 100 mA draw)
RST	RESET- for processor. Pull low to initiate a board reset

There are three ways of powering the logic circuits for system, which can simplify selection of a power supply to use in driving the system. There is a 3-position jumper positioned just beside the PWR power connector, as shown below.

The positions are labeled as:

- USB (short top 2 pins) – Power is provided by the USB system.
- 5V (short center 2 pins) – You are providing fully regulated 5 volts.
- >6.5 (short bottom 2 pins) – You are providing regulated 6.5 to 15 volts.

In both methods of powering our board with your own power supply, 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!***

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, and set the PWR jumper to the ">6.5" position. 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 +Vc and nearest GND connector, and set the PWR jumper to the "5V" position. Your 5V source will also be presented at the +5V/GND connector.
3. If you wish to power the board off of the USB system, place the PWR jumper into the USB position. Do not draw more than 100mA from the +5V pin when this option is selected. Power will be present as long as the USB system from the computer is connected and "awake" – if the computer is placed into its sleep mode, then the board will (mostly) go to sleep, and the +5V will be turned off. **Always unplug the USB connector from the board at any time that you change any wire on the board! As long as the unit is plugged in, it is powered!**

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
IO7	+128	Generic TTL I/O 7
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 TTL-Serial and generic TTL I/O lines.

The IO7 and IO6 signals are used to connect the board to the SI and SO lines (respectively) of any other motor controller from our company. Through use of the serial routing capabilities of the firmware (see "**Error! Reference source not found.**" starting on page **Error! Bookmark not defined.**), a 'child' board can be "daisy-chained" to receive serial data over the same connection which is used to send serial data to the SD4D board. IO7 is to be connected to the child board's SI line, while IO6 is to be connected to the child board's SO line.

The remaining signals (IO5 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	Register Bit	Value when setting as pulse generation or other output
WDR	7	+128
WST	6	+64
XDR	5	+32
XST	4	+16
YDR	3	+8
YST	2	+4
ZDR	1	+2
ZST	0	+1
GND		Logic signal ground

This connector gives access to the actual step-and-direction signals as generated by the board. For the SD4D01, all of the outputs are normal TTL levels (0-0.4 volts for low, 4.5 to 5 volts for high). For the SD4D02 ("G" release), these pins are actually outputs of an on-board ULN2803 buffer driver, for compatibility with high-current devices.

On the original SD4D version 1 release, the signals can drive no more than 5 milliamps of current each; 1 mA each is much preferred. If you are going to need to drive more than 5 mA, then you will have to provide your own buffer circuit to provide the extra power needed.

On all of the other SD4D releases, the outputs are normally buffered through a ULN2803 driver. This gives you adequate current to operate higher current outputs.