Peter Norberg Consulting, Inc.

Professional Solutions to Professional Problems

P.O. Box 10987 Ferguson, MO 63135-0987 (314) 521-8808

PRELIMINARY AND INCOMPLETE Information and Instruction Manual for FENCMCT Firmware and the FENC series of Encoder Controllers

By

Peter Norberg Consulting, Inc.

Matches FENCMCT Firmware Revision 5.08 (Beta Release)

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 reported to us, 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 FENC artwork revisions 1 and 2. The firmware release described is FENCMCT version 5.08 Beta. The manual version shown on the front page normally has the same value as the associated FENCMCT 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**
5.0	January 9, 2014	First manual release, ALPHA level code (pre-beta release)
5.01	January 16, 2014	Still alpha code; adds 'T' command, extends 'M' command
5.03	January 22, 2014	Still alpha code; finalizes 'T' command with new 'S' command, adds report of 'S' setting
5.05	January 24, 2014	Promoted to Beta code. Adds '-16?' query to access the encoder value associated with the first pulse generated as a result of encoder MAXCNT processing. Corrected potential issue related to maintaining the encoder location.
5.06	January 27, 2014	Adds "-17?" query, to report the 'm' setting. Correct the save/restore settings 'e' commands to actually work, and added forcing of all TTL I/O commands to conform to legal bit mask values. Code is still considered to be Beta level.
5.08	February 11, 2014	Changed encoder-based pulse generation code to be unidirectional (i.e., the code caculates the direction at the time of issuing the 'P' command). This also changes 'm' to specify the minimum amount by which the target specified by 'P' must differ from the current location to avoid being adjusted by the 'M' value.

Product Safety Warnings

The FENC 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 only partially protected against static electricity (ESD). 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 **FENC** encoder monitor controller from Peter Norberg Consulting, Inc., has the following general performance specifications:

	FENC
Logic supply voltage (+5V)	5V or 6.5 to 15V
Average Current draw	150 mA
Board size	3" x 1.75"

The FENCMCT firmware is designed to allow simultaneous monitoring of up to 1 high speed phase encoder, generation of 3 discrete pulse output signals, as well as generic TTL input and output. It also supports 'daisy chaining' of Peter Norberg Consulting, Inc. compatible products, through use of its TTL-Serial based SI2/SO2 pins.

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):

*257t	- Configure encoder to be active, and to operate in X2 mode. Also configures pulse generator CNT0 to be 'attached' to encoder events related to MAXCNT pulse over or underflow (i.e., generate a pulse every MAXCNT encoder events)
*1B	- Select pulse counter CNT0
*1g	- pulse counter CNT0 uses the PWM1L output
*5W	– Set both the 'high' and "low" pulse widths to be 25 microseconds
*0D	- Preload all TTL output ports to be 0
*1d	- Set PWM1L pin as an output
*100m	–Force encoder delta from currently location to be a minimum of 100 counts, to allow accurate targeting of start pulse location
*1000M	– Set to generate a pulse every 1000 encoder pulses
*0=	- Set current encoder logical locati0n to be 0. Due to above '10m', the first encoder event on forward counts would thus occur at event 990
*100G	- Generate at most 100 pulses in response to encoder events

Short Feature Summary

- Up to three separate pulse generators may be running concurrently, each operating any combination of the TTL output ports.
- 1 high speed phase-encoder may be simultaneously monitored
- Up to 8,000,000 encoder 'edges'/second may potentially be counted. This is equivalent to a pulse rate of about 2,000,000 pulses per second on one of the encoder input channels.
- 20 TTL I/O lines are available. Most of the lines 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).
- 4 of the inputs are wired such that they can either be used as TTL inputs or as differential inputs (based on where you place the wires). Three of these signals are the encoder inputs (QEA, QEB and the INDEX pulse), with a fourth being available as a generic input.
- Runs off of a single user-provided 6.5 to 15 volt regulated DC power supply, a single user-provided 5 volt regulated DC power supply, or USB power.
- Communication is via USB or TTL-Serial

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 microprocessor 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 DSPIC chip. The fan should provide about 6-10 CFM of air flow. If the fan is mounted facing down at the top of the board (which cools the DSPIC 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

The board normally acts at power on (or reset) as if the following serial commands have been given (note that all of these commands can be overridden by you as new defaults through use of the 'memorize settings' command, "123456789e'):

- **7M** – Select all three pulse generators for actions
- **0G** – Stop all pulse generators
- **0**= – Define encoder to be at location 0
- **0d –** All I/O lines are inputs
- **0D –** All programmable outputs are set to 0.
- **1V** – Set <CR><LF> sent at start of new command, no transmission delay time
- **9600b** – Set communications baud rate to 9600 baud
- **65535M** – Set MAXCNT to maximum legal value
- **100m** – Set encoder delta minimum to 100 counts
- **128t** – SI2/SO2 are set as TTL-Serial ports, encoder is off
- **1W** – Pulse widths are 5 microseconds high, 5 microseconds low
- **1x** – Start pulse delay (first pulse only) is 5 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 FENC 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 FENC 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 encoder) to install the drivers: just the FENC 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 FENC, 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 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 16 TTL I/O signals, 4 TTL input-only signals and 4 differential input signals that multiplex with those four TTL input-only signals.

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.

TTL Input Voltage Levels: Schmitt-Triggered 5 Volt TTL

All TTL input signals are treated as Schmitt-triggered 5 Volt TTL levels. This means that a logic “0” is generated at any time that the input voltage drops below about .6 volts, and a logic "1" is generated any time that the input voltage then rises above about 4.2 volts.

Most of the TTL I/O signals "float"; that is to say, they are not pulled up or down through use of termination resistors. The signals labeled "IO0" through "IO4" are pulled up to +5 via 10K pullups, and the four signals RB2, IDX, QEA and QEB are cross-connected via 10K isolation resistors to the output of the differential receiver attached to the differential versions of the RB2, IDX, QEA and QEB signals. This allows you to either connect your signals to the differential inputs or the TTL inputs of the matching signal, but not both.

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. You need to keep your current demands down to at most 5 mA per line.

Additionally, if you actually do drive signals at noticeable current levels, the DSPIC chip will get warm, and may get hot. You may find that you have to cool the board if the DSPIC seems to be running too hot.

If you exceed these limits, the DSPIC 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 any pulse generation sequence, 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.

EXIO connector has inputs with 10K pullups, lines IO0 to IO3

Lines IO0 through IO3 are normally shipped with 10K pullup resistors installed, and are usually used to trigger special events in the firmware. If they are to be used as outputs (see the 'd' command), you may elect to special-order your board from the factory with this 10K pullup removed.

PWM signals P1x, P2x and P3x

The six PWM signals are expected to be used as pulse outputs from the board, although the firmware defaults to using them as inputs to avoid driving a signal incorrectly. They are the pins to which CNTx outputs may be directed on pulse generation, and may be used both as generic TTL input and outputs as needed by your application. No pullups are attached to these signals.

IO connector contains TTL-Serial signals SI, SO, SI2 and SO2

The IO connector contains the two pairs of TTL-serial signals. SI and SO are the raw TTL-Serial inputs to the board, while SI2 and SO2 are the generic 'daisy-chain' serial for use in communicating with another Peter Norberg Consulting, Inc. board on the system.

SI and SO and usually are driven by the USBToTTL chip that is mounted on the unit. You may override the USB system by connecting your own TTL-serial signals to these pins: SI is the board-level TTL-Serial input, while SO is the board level serial output. See the 'b' command for control of the baud rate used; by default, the board uses 9600 baud, 1 start bit, 8 data bits, 1 stop bit, and no parity.

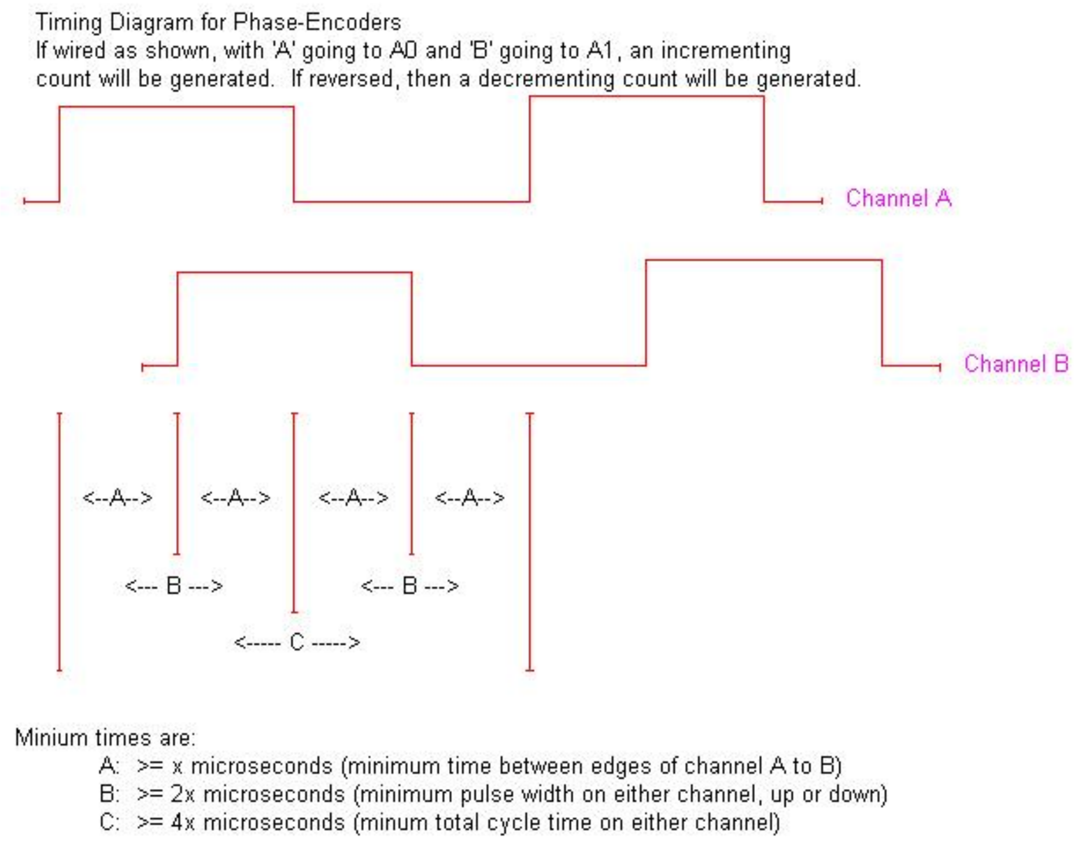
The SI2 and SO2 signals are used via the serial routing commands ("{0}" and "<", ">"). They allow multiple boards to be connected to your computer via one serial connection. You may use the 't' command to redefine these signals as generic TTL I/O lines; otherwise, SI2 is TTL-Serial input to the board while SO2 is TTL-Serial output from the board.

DIFF connector contains differential inputs for use by the encoder

The DIFF connector includes standard differential inputs for the QEA and QEB quadrature inputs, the IDX index pulse, and one extra generic RB2 input for use as needed by your application. After being converted to TTL by a differential receiver, these four inputs are connected to the QEA, QEB, IDX and RB2 pins on the TTLIO connector via a set of 10K isolation resistors. This means that, for each of those four signals, you may either connect a differential set of signal sources (to the differential version of the pins) or a TTL set of signal sources (to the TTLIO version of the pins), but not to both.

Encoder Connections

One high speed phase-encoded positional encoder may be connected to the QEA and QEB inputs on the board (either differential or TTL). In each case, the code has the following signal requirements:



Timing Diagram for Phase-Encoders
If wired as shown, with 'A' going to A0 and 'B' going to A1, an incrementing count will be generated. If reversed, then a decrementing count will be generated.

Minium times are:
A: >= x microseconds (minimum time between edges of channel A to B)
B: >= 2x microseconds (minimum pulse width on either channel, up or down)
C: >= 4x microseconds (minum total cycle time on either channel)

The current hardware requires at least 1/8th of a micosecond as the 'x' value shown above. If you enable filtering of the inputs (see the 't' command), this timing requirement goes up as the filter goes up.

PWR connector provides power for the board

The PWR connector gives you two of the three ways of providing board power.

You may connect your power using exactly one of the following methods:

1. Connect a 6.5 to 15 volt regulated DC source to the +Vc/GND pin pair, and remove the U5V jumper near the PWR connector.
2. Connect a 5 volt regulated DC source to the +5/GND pin pair, and remove the U5V jumper near the PWR connector.
3. Connect nothing to the PWR connector, and install the U5V jumper near the PWR connector. The board will be powered off of your USB connection.

NEVER connect a power supply to the PWR connector while having the U5V jumper installed. You **will** damage the board, and you may damage your computer!

Serial Operation

The USB serial control of the system allows for full access to all internal features of the system. It operates at almost any legal baud rate from 2400 to 115,200, 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. You may use the 'b' command to select the baud rate: it will accept almost any rate, although only a few will be 'perfect'. The suggested rates to use would be 2400, 4800, 9600, 19200, 38400, 57600 or 115200.

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,

```
1000M
```

would mean "set the MAXCNT counter to 1000"

```
2000G
```

would mean "Generate 2000 pulses on the currently selected counters".

The firmware actually recognizes and responds each new command upon completion of receipt of the stop bit of the received character. This means that the command normally starts being processed within 16 microseconds of completion of the character bit stream. In most designs, this will not be a problem; however, since all commands issue an '*' upon completion, and they can also (by default) issue a <CR><LF> pair before starting, it is quite possible to start receiving data pertaining to the command before the sending processor is ready for it! In microprocessor, non-buffering designs (such as with the Parallax, Inc.tm Basic Stamp tm series of boards), this can be a significant issue. The firmware handles this via a configurable option in the 'V' command. If enabled, the code will pause for about 1 millisecond after receipt of a new command character; for the Basic Stamptm this is quite sufficient for it to switch from send mode to receive mode.

Routing Serial to 'Child' Boards

The FENC series of boards supports optional "Serial Routing" of data to and from other boards via the "SI2" and "SO2" TTL-Serial connectors. This permits "daisy-chaining" of the boards, so that more than one board may be easily operated off of one serial port. Additionally, FENC supports "binary" serial transmissions – that is to say, it allows communication with a "child" board which includes non-visible characters, thus permitting communication with other products not supplied by Peter Norberg Consulting, Inc.

Connecting your 'Child Board'

The SI2 and SO2 signals on the FENC are normally used as the TTL-Serial connections to the child board. These are NOT RS232-level signals; instead, they are standard TTL signals, 0 to 5 volt in value. The signals are:

- SI2: TTL-Serial Input received from child board
- SO2: TTL-Serial Output sent to child board

You also need to make certain that both boards have the same ground potential: they both need to have their logic supplies run off of the same power supply. Ground should be done via a "star" ground, which means that all grounds are tied to a common point (NOT "daisy-chained"!).

Routing the Serial Data

At power on, the board is configured to directly interpret and execute all incoming serial data, and to leave the SI2/SO2 serial echo lines idle. This mode means that the board is under control of the serial data stream from the host. There are two methods of exiting this mode, and switching to serial-routing operation (wherein data is routed from the serial port to and from the SI2/SO2 pins as needed).

One method, the "Binary Route" technique, allows single-character entry into route mode, and is the preferred method when controlling only one child board or when controlling a child board which is not provided by Peter Norberg Consulting, Inc. (and therefore does not match our normal communications protocol).

The other method, the "SerRoute compatible" technique, allows the FENC board to operate as both a router and a controller within a subset of the rules of the "SerRoute" system (see the separate "SerRoute" manual for complete SerRoute specifications – FENCMCT version 5.0 supports just the '{xxx}' style of routing).

'<' – Start Binary Route

The '<' character immediately starts routing all serial data to and from the child board. Except for the route command characters ('<', '>', '{', '}') and the ESC character ('\'), all characters from the serial port are sent unchanged to the SO2 child serial port without any other interpretation by the firmware.

This mode is the preferred method when operating any board as a child that is not a Peter Norberg Consulting, Inc. product. It permits transmission of binary data, and avoids any issues of sending spurious route-related commands (such as the '{' "start route selection" character).

Please note that, when routing in the binary route mode, you must precede certain characters with the escape character '\' in order to prevent the firmware from interpreting those characters as route commands. This means that you should precede any of '<', '>', '{', '}', or '\' with a '\' ("escape it"), so that it will be passed on unchanged to the child board.

'>' – Stop Routing

The '>' character immediately stops any routing which is taking place. This is normally used to exit the routing mode, and return to "talking" to the main board.

Note that if the '>' character is preceded by the '\' escape character, the '>' is just sent on to the child (assuming that routing is enabled), and route mode is not canceled.

'{xxx}' – Select SerRoute compatible Route Mode

This sequence allows routing using a method which is compatible with the SerRoute firmware. The 'xxx' values are used to select which "route" is to be enabled, while the '{' and '}' are used to bracket the route address. Please see the 'SerRoute' manual for a full explanation of the routing rules: from the point of view of FENC, however, there are only 3 important SerRoute-compatible route modes, as selected by the first character in the 'xxx' value.

x	Description
Empty	If just '{}' is sent, then the current board is selected for all serial. This is exactly equivalent to receipt of the '>' character, above.
'0'-'8'	All serial data is routed to pin SO2, and all serial data seen on pin SI2 is routed back to the host.
'9'	Both this board and the child board receive all following serial data, while the serial data sent back to the host is strictly that from this board. Used to execute commands simultaneously on both boards.

In reality, when interpreting the '{xxx}' style of input, the code operates as follows:

1. When the '{' is seen, the code sets a flag for 'the next character tells us how to route all of the following characters'.
2. When the first 'x' character is seen, if it is not '}' or '0'-'9', the route command is cancelled (illegal sequence), and the 'x' character is interpreted as a new board command. I.e., '{g' is interpreted as 'g' (although technically speaking, this form of command is not proper).
3. If the first 'x' character is '}', then routing is terminated. All following characters are treated as normal data to the board, and no further activity occurs relative to the SI2/SO2 ports (aside from completing any pending serial transmission).
4. If the first 'x' character is '0' to '8', then all future serial data is routed to the child board via the SI2 and SO2 lines. The '{' "route selection" character is passed on to the child, to allow for nested route control.

5. If the first ‘x’ character is ‘9’, then (as with ‘0’ to ‘8’) all future serial data is replicated to the IO7 output line. However, it is also executed locally, thus placing the board into the mode of ‘broadcast’. The ‘{‘ “route selection” character is passed on to the child, to allow for nested route control.

For example, if we have 3 FENC boards with FENCMCT firmware “daisy chained” such that:

- the first board “talks” via its USB port to the host,
- the second board has its SI line connected to the first board’s SO2 line, and its SO line to the first board’s SI2 line, and
- the third board has its SI line connected to the second board’s SO2 line, and its SO line to the second board’s SI2 line, giving us:

First Board

→ Second Board

→ Third Board

We would address the first board with:

{}

The second with:

{0}

The third with:

{00}

And all three at once with:

{99}

Please refer to the “SerRoute” manual for many more examples of this technique.

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

0-9, +, - – Generate a new VALUE as the parameter for all FOLLOWING commands
L – Latch Report: Report encoder overrun and ‘restart’ latches, reset latches to 0
V – Verbose mode command synchronization
! – RESET – all values cleared. Duplicates Power-On Conditions!
? – Report status

I/O Port direction control, direct set and clear of outputs, I/O Trigger commands

d – Define I/O direction for I/O ports
C – Set selected outputs high (sets the output port bits to 1)
U – Set selected outputs low (set the output port bits to 0)
D – Set all outputs high or low as specified in the command
S – Define bits used on Trigger state change command
T – Wait for change in input bits as defined by S and T commands

Pulse Generation Commands

d – Define I/O direction for I/O ports
B – Select pulse generators to access for the subsequent commands
g – Specify which I/O register bits are toggled by pulse current pulse generator(s)
W – Set pulse Width after first edge (high time)
w – Set pulse width after second edge (low time)
xM – Set MAXCNT value as used by encoder system for pulse control
xm – Set minimum encoder delta from current location when 'P' is requested
xP – Set first Pulse location for encoder-generated pulses
G – Generate x pulses on current generator(s)
t – Configure encoder usage and connection to pulse generators
? – Report pulse generation states

Encoder Commands

D – Define I/O direction for I/O ports
= Assign current encoder values
t – Configure encoder usage and connection to pulse generators
? – Report encoder values

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	GEN0
1	+2	GEN1
2	+4	GEN2

A total value of 0 is treated as 7; that is to say, '0B' selects all of the pulse generators.

xa – Read A/D channel

This command allows you to read one of the four available 10 bit A/D input channels from the TTLIO connector, as requested by the parameter (x). The given input must already be configured as a TTL input through use of the 'd' command in order for this function to not report a -1 result.

The firmware takes 16 samples on the requested channel, and reports the average of those samples. The channel numbers match the labels on the TTLIO connector, so RB0, RB1, RB8 and RB7 are available. RB2, RB3 (IDX), RB4 (QEA) and RB5 (QEB) will not provide reliable analog data, due to how they are cross-wired to the DIFF input signals.

```
0a – RB0
1a – RB1
6a – RB6
7a – RB7
```

The report consists of data in the following form:

a,#CH,#AD

where:

'a' is the report type; in this case, 'analog data report'

'#CH' is the channel being reported

'#AD' is the actual 10 bit (0 to 1023) reading for the channel

For example,

0a

Could report

a,0,345

which would mean 'the signal on RB0 had an A/D value of 345'.

A reading of 1023 nominally maps into 5 volts. In reality, it maps into whatever voltage is present on the 5 volt bus on the FENC board. If you are operating the board purely off of USB, this board "5 Volts" may be as low as 4.6 volts.

xb – Select the communications baud rate

The '**b**' command is used to change the communications baud rate from its current value to a requested new value. The factory default setting is 9600 baud, or

9600b

The code takes the rate that you request, and calculates the closest rate to that which it is capable of supporting. It then echoes that rate back, in a response formatted as:

b,#

where '#' is the real baud rate that the code will be able to use. After it completes sending of the final '*' in its response, it will then reset itself to operate at the newly specified rate.

For example, to set the firmware to communicate at 19,200 baud, you would send:

19200b

and you would get the response:

b,19200

*

After you receive the '*', you would then reset your serial system to operate at 19,200 baud.

Suggested values for the baud (which are used exactly) are:

```
2400
4800
9600
19200
38400
57600
115200
```

Many other rates are supported; however, the above list contains the most commonly used values.

Please be forewarned! If you use a non-standard value, or a value above 19,200, then our Stepperboard class libraries (both the Active-X and the Visual Studio .NET systems) may not be able to 'find' the board using an automatic scan. You will need to directly tell the class library what baud rate and what port to use for your application.

If you attempt to set the baud rate to an unsupported value, and ignore the response which contains the real value used, you may not be able to talk to the board. Simply power it off and back on: it will revert to the baud rate that was active the last time that you performed the special '123456789e' save state command, or to the 9600 baud factory default if you have never saved the firmware state. If you have saved a non-standard baud rate using the save-state command and you forget what you used as the baud rate, you will have to perform the factory-reset action, as described elsewhere in this manual, in order to revert the firmware to a known standard state.

xC – Set selected TTL outputs HIGH

This command allows you to selectively set a subset of the bits on the TTL outputs 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	PWM1L
1	+2	PWM1H
2	+4	PWM2L
3	+8	PWM2H
4	+16	PWM3L
5	+32	PWM3H
6	+64	<no connect>
7	+128	<no connect>
8	+256	IO0
9	+512	IO1
10	+1024	IO2
11	+2048	IO3
12	+4096	<no connect>
13	+8192	SO2
14	+16384	SI2
15	+32768	<no connect>
16	+65536	RB0
17	+131072	RB1
18	+262144	<RB2, but not available for output>
19	+524288	<IDX, but not available for output>
20	+1048576	<QEA, but not available for output>
21	+2097152	<QEB, but not available for output>
22	+4194304	RB6
23	+8388608	RB7

For example, to set IO2 and PWM1L to '1' while leaving the rest unchanged, send the command:

```
1025C
```

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 'd' command.

xd – Define I/O Port Directions

This command is used to define the I/O directions for all of the programmable I/O ports.

Setting a bit high (to 1) causes that bit of that port to be set as an output, setting it to low causes that bit to be an input.

The encoding matches that of the various TTL I/O commands:

Bit	***Value***	***Signal***
0	+1	PWM1L
1	+2	PWM1H
2	+4	PWM2L
3	+8	PWM2H
4	+16	PWM3L
5	+32	PWM3H
6	+64	<no connect>
7	+128	<no connect>
8	+256	IO0
9	+512	IO1
10	+1024	IO2
11	+2048	IO3
12	+4096	<no connect>
13	+8192	SO2
14	+16384	SI2
15	+32768	<no connect>
16	+65536	RB0
17	+131072	RB1
18	+262144	<RB2, but not available for output>
19	+524288	<IDX, but not available for output>
20	+1048576	<QEA, but not available for output>
21	+2097152	<QEB, but not available for output>
22	+4194304	RB6
23	+8388608	RB7

For example, to define PWM1L through PWM3H as outputs, and the rest as inputs, issue the command:

```
63d
```

Please note that IO0 through IO3 have pullup resistors installed. Defining those ports as outputs will heat the microprocessor somewhat – you may elect to order the board with those pullup resistors removed.

xD – Set all TTL outputs high (1) or low (0) as requested.

This sets all outputs values shown:

Bit	***Value***	***Signal***
0	+1	PWM1L
1	+2	PWM1H
2	+4	PWM2L
3	+8	PWM2H
4	+16	PWM3L
5	+32	PWM3H
6	+64	<no connect>
7	+128	<no connect>
8	+256	IO0
9	+512	IO1
10	+1024	IO2
11	+2048	IO3
12	+4096	<no connect>
13	+8192	SO2
14	+16384	SI2
15	+32768	<no connect>
16	+65536	RB0
17	+131072	RB1
18	+262144	<RB2, but not available for output>
19	+524288	<IDX, but not available for output>
20	+1048576	<QEA, but not available for output>
21	+2097152	<QEB, but not available for output>
22	+4194304	RB6
23	+8388608	RB7

For example

7D

will set outputs PWM1L, PWM1H and PWM2L to be high, and all other output bits to low.

xe – Write EEProm data

The 'e' command is used to rewrite the EEProm data on the board. As of version 5.23 of the firmware, 'e' is ignored if you fail to provide a parameter (i.e., 'e' alone results in the board ignoring the command, while '0e' causes the normal 'e' operation with a parameter of '0').

The data value passed is used to control what kind of write action is to be performed, and is range-based. The data may be written and erased at least 100,000 times.

Value Range	Use
0-65535	Write indicated value into the current user-data 'eeprom' write address, then increment that address for the following write. This allows access to 32 words (64 bytes) of user-programmable memory
65536-65567	Specify next write address for user data write (above)
123456789	Write control portion of EEProm with current state of system. This saves all savable parameters (such as ramp rate, run rate, microstep size, I/O configuration, baud rates and so on) for use on the next reset of the board.
987654321	Erase the complete EEProm, restoring the system to factory defaults on the next boot.
Other	Illegal – firmware generates a verbose error response

Writing User data

The firmware supports 32 16-bit words (64 bytes) of user programmable EEProm. After a reset, the next EEProm write address is set to the first word of user data (User data address 0); from then on, each successive 'e' command with a data value of 0 to 65535 will write to the current user data address, and will then increment that address for the next write. If the new address would exceed 31, it stays at location 31.

To read that data back, use the '?' command with a value of 16384+user.address. The code will reset the user-data access location to the desired address and report the current contents. It will also leave the pointer at that location, so that the next user data write will overwrite the location.

Alternatively, you may specify the next write address by using an 'e' command with a data value of 65536+user.address. This simply resets the current address to the desired value.

For example: to read and then update user data location 3, you could issue the commands:

```
16387?
     The board could respond with "X,16387,10" and an '*'
25e
     The board would respond with "e,25" (which confirms the write)
16387?
     The board would respond with "X,16387,25"
```

To simply write a '5423' to location 15, you could issue the commands:

```
65551e
     The board would respond with just an '*'
5423e
     The board would respond with 'e,5423'
```

Note that the code is intelligent enough to not actually erase and rewrite the data if the new value matches the old value.

Warning! You must wait for the microprocessor to respond with the standard command completion character '*' before transmitting any more commands! The firmware automatically reduces the clock rate to about 7.4 MHz while updating the EEProm, which will cause loss of serial data if any data is sent to the board during this event. Also, pulse generation and encoder actions are temporarily suspended during any 'e' command, to avoid corruption of the EEProm.

Saving the current firmware settings

The current firmware settings may all be saved by issuing the command:

```
123456789e
```

This saves everything that is capable of being saved: this includes such items as the ramp rates, start/stop rates, A/D control values, and all other key I/O port information. Items that are not saved are transient data such as current I/O port contents and current encoder locations.

The new settings are used the next time the board is powered on, and the next time that a board reset (either soft via the '!' command or hard via the RST input) is issued.

Restoring firmware settings to Factory Default values

The firmware EEProm may be reset to factory defaults by two methods.

1. Issue the '987654321e' command.
2. Do an "EEPROM RESET" hardware strap startup as is described below.

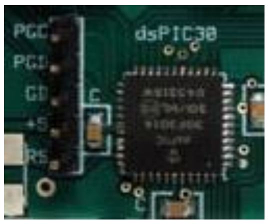
Both methods erase the EEProm flash memory, including any data that you have saved using the above "user data" write commands. This sets all of the memory to have a value of 65535, which is used by the firmware to tell it that the memory is uninitialized.

The first method purely erases the memory. It does not otherwise affect the current state of the board – that is to say, your current baud rates, start/stop rates and so on will remain unchanged until you reset the board (at which point the factory default settings become active).

The second method ("EEPROM RESET") is your emergency recovery method to allow you to regain control of a board which has unknown settings and a saved baud rate that cannot be discovered using our standard tools. The technique is:

1. Power off the board
2. Disconnect everything from the board except for the power connection and the USB connection (for safety)
3. Use a standard 0.1" shunt to jumper the "PGD" and "PGC" pins together that are on the 5 pin header on the top of the board (See below)
4. Power the board on
5. Confirm that you can 'talk' to the board (it will be communicating at 9600 baud); this is most easily done by running the 'TestSerialPorts' application
6. Power the board off
7. Remove the shunt from PGD and PGC
8. Reconnect your encoder and wires as needed

The board should now be restored to its factory-default settings. If you still cannot 'talk' to the board, please contact us for further instructions.



The top 2 pins are PGD and PGC
(which you need to short together
In order to perform the reset action)

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’, 'x' 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’, 'x' and ‘g’ commands. 'x' controls the delay before the start of the first pulse, ‘W’ controls the time for the first 'edge' transition of the pulse, while ‘w’ controls the time after the second edge transition (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.

For example, to generate one 25 millisecond pulse on the PWM1L signal, you could issue the following sequence:

1B	Select pulse generator 0
1g	Define that generator to be accessing only the PWM1L signal
5000W	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 ‘5000W’ is exactly equivalent to the sequence:

```
5000W
5000w
1x
```

Note that pulse width times are in units of 5 microseconds.

When 'G' is requested on a pulse generation attached to the encoder MAXCNT processing system, a special flag is set that tells the code to note the exact encoder location associated with the FIRST pulse generated by that sequence. This allows you to obtain that value (via the "-16?" query), and use it to calculate the exact locations of each pulse that would be generated by that system.

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

‘g’ is used to select which of the 20 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 the low 6 bits used by the TTL output commands (such as ‘C’), and are as shown in the following table.

Bit	***Value***	***Signal***
0	+1	PWM1L
1	+2	PWM1H
2	+4	PWM2L
3	+8	PWM2H
4	+16	PWM3L
5	+32	PWM3H

Therefore, to connect pulses to the ‘PWM1H' 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 ‘d’ 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– 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	
1	+2	
2	+4	
3	+8	If set, the encoder has recorded a large amount of motion that is in the reverse direction from that requested by the current 'P' based encoder-driven pulse generation. The system has disabled pulse generation until a new 'P' is requested.
4	+16	System power-on or reset ("!") has occurred
5	+32	If set, then the board is running from a precision clock source
6	+64	If set, the RESET strap is installed. The board will not operate correctly until the RESET strap is removed!
7	+128	

For example, after initial power on,

```
L
```

Would report

```
L,16
*
```

xM – Set MAXCNT value as used by encoder system for pulse control

The 'M' command interacts with the 't' command to give you control over how many encoder edge events (counts) are needed to trigger a counter/interrupt update action. Its most common use is to specify the number of encoder pulses that is desired between CNT0 pulse outputs.

M may not be set to a value that would generate interrupts too frequently: if the value requested would generate interrupts every 10 microseconds or so, it is likely that the system will not work correctly.

The firmware currently accepts M values in the range of 10 to 1,000,000,000 counts. If MAXCNT is set to a value above 65535, the actual number used internally by the hardware is going to vary between 32768 and 65535, depending on the internal software state of pulse generation.

xm – Set minimum encoder delta from current location when 'P' is requested

The 'm' command is used to specify the minimum distance from the current location to the first pulse generated from an encoder-driven pulse generation sequence. It has a legal range of 1 to 65535 counts, with a value of 100 being assigned (the default) if a 0 is requested. If a 'P' first-pulse location is specified whose distance from the current encoder-based location is less than the 'm' value, the first-pulse location will be adjusted by the 'M' (encoder period) value, to allow for enough time for the encoder system to stabilize. For example:

```
    10000M  - Request an encoder pulse period of 10,000 encoder pulses/generated pulse
    100m    - Set that the 'P' target must be at least 100 counts from current
    0=      - Define current location as 0

If you now issued:

    500P

the target location for the first generated pulse would be 500.   If you instead
issued:

    50P

the target location for the first generated pulse would be 10050, since the current
location (0) is less than 100 steps from the requested target (50).
```

Discussion: The firmware can only generate encoder-driven pulses based on interrupt synchronization with the hardware. To get to a given starting point, the hardware must be told to interrupt every 'x' pulses, where 'x' is the distance from the current location to the first-pulse target location. If this happens just once, there are no issues with any non-0 value for this synchronization. However, it is quite possible that the motor could start moving in the reverse direction before the initial target location is reached. If this happens, the firmware will track the incorrect motion so as to correctly trigger once the motion goes back to being the 'correct' location. Due to hardware constraints, this has to be done by counting the number of interrupts that are in the 'wrong' direction, with each interrupt being that 'x' encoder pulses in time. The hardware has a timing constraint that prevents reliable encoder tracking if more than 1 encoder-driven interrupt can occur within 10 microseconds. If you move in the 'wrong' direction with a small distance specified (for example, 'm' is 1, with the 'wrong distance' traveled being 20 counts), you could get 20 attempts at an interrupt within that 10 microsecond interval. That cannot happen, so the synchronization between the firmware and the hardware would be lost, which would mean that the reported encoder location would no longer be valid.

If you have a very clean, jitter-free system, then smaller 'm' values may be used. However, the default '100m' is the safest minimum for normal operation.

xP – Set first Pulse location for encoder-generated pulses

'P' is used to specify the exact encoder-based location for the first pulse to be generated until an encoder-based pulse generation sequence. The firmware looks at the current encoder-based location, and calculates the distance and direction from that location to the requested 'P' location. The direction controls the required direction of motion for all subsequently generated pulses, while the distance controls when the first pulse is generated.

'P' interacts with the 'M' and 'm' values in regards to the real first generated pulse location. 'm' is used to specify the minimum delta from the current location to that first generated pulse location (usually, this minimum is set to 100 encoder counts). If the requested target is closer to the current location than the minimum, then the 'M' value (the pulse generation period) is added to the distance to calculate a corrected first pulse location.

Please see the 'm' command (above) for a discussion on this.

xS – Set input mask to use on 'T'rigger event processing

This command defines the bit mask used to control which input bits control the trigger, while 'T' controls what kind of trigger is used, and actually waits for the trigger event.

The 'x' parameter passed to the 'S' command describes which I/O lines are to be monitored for a change when the 'T' command is issued. When the type of change that is specified by 'T' is detected, the 'T' command terminates and sends back a report that tells you which lines caused it to respond.

The encoding matches that of the various TTL I/O commands:

Bit	***Value***	***Signal***
0	+1	PWM1L
1	+2	PWM1H
2	+4	PWM2L
3	+8	PWM2H
4	+16	PWM3L
5	+32	PWM3H
6	+64	<no connect>
7	+128	<no connect>
8	+256	IO0
9	+512	IO1
10	+1024	IO2
11	+2048	IO3
12	+4096	<no connect>
13	+8192	SO2
14	+16384	SI2
15	+32768	<no connect>
16	+65536	RB0
17	+131072	RB1
18	+262144	RB2
19	+524288	IDX
20	+1048576	QEA
21	+2097152	QEB
22	+4194304	RB6
23	+8388608	RB7
24	+16777216	CNT0 busy bit
25	+33554432	CNT1 busy bit
26	+67108864	CNT2 busy bit

If you specify a value of '0' in the 'S' command, it resets it to be an 'all bits' check. The firmware automatically removes all illegal bits from the test (for example, the <no connect> bits are automatically ignored by the system).

xT – Wait for TTL I/O trigger event

This command is the action mode of the 'S', 'T' trigger pair. 'S' is used to set the mask that controls which input bits control the trigger, while 'T' controls what kind of trigger is used, and actually waits for the trigger event.

This command will NEVER terminate if no I/O bits change, but it can be aborted by the board receiving any other command (which will be processed normally).

The 'x' parameter to the command describes how the input ports are monitored for the trigger value. There are 3 basic modes of trigger:

- Trigger on any/all selected bits low
- Trigger on any/all selected bits high
- Trigger on any/all selected bits change of value

The 'any/all' mode is selected by one of the bits in the 'x' parameter: if it is set, then all of the input bits that are selected by the 'S' mask must match the requested state (i.e., all high, all low, or all change). If it is low, then any of the bits matching the selected state will complete the action.

The x parameter is encoded as:

Bits	***Value***	***Trigger Mode***
0-1	0	Any/all selected input bits low
	1	Any/all selected input bits high
	2	Any/all selected input bits change
	3	<reserved: will always immediately finish>
2	+4	If clear, the mode of the test is 'Any'. If set, the mode of the test is 'All'.

For example, to wait on any of the IO0 through IO3 bits going high, you would set the mask to

```
3840S
```

You would then issue the 'wait' command based on any of those bits being high:

```
1T
```

The firmware would then wait until any of the IO0 through IO3 lines became high, and would report which ones were high:

```
T,512
*
```

(the above report would mean IO1 was detected as being high).

As another example, to wait on all pulse-generation counters to complete their counts, you would arm the system with the 'S' command of:

```
117440512S
```

You would then issue the 'wait' command based on all of those bits being low:

```
4T
```

After all 3 lines go low (pulse generation complete), the firmware would report

```
T,117440512
*
```

If the command is aborted by another command, it will respond with

```
T,0
*
```

(which may be dropped from being transmitted, depending on your 'V' setting).

xt – Configure encoder-related I/O port actions, serial daisy chaining, and counter usage

The 't' command is used to configure the board's encoder, serial daisy chaining via the SI2/SO2 lines, and the extended options available from the 3 board pulse generators.

The command is encoded as:

Bit	*Value*	*Signal*
0	+1	ENCODER input enable. If set, the encoder is turned on, and transitions on the QEA and QEB inputs will generate counts.
1	+2	Encoder scale mode: 0 means x2, 1 means x4.
2-4		Digital filter to use on the encoder, if bit 5 is set.
		4 3 2

	+0	0 0 0 1:1 filter
	+4	0 0 1 1:2 filter
	+8	0 1 0 1:4 filter
	+12	0 1 1 1:16 filter
	+16	1 0 0 1:32 filter
	+20	1 0 1 1:64 filter
	+24	1 1 0 1:128 filter
	+28	1 1 1 1:256 filter
5	+32	If set, enable the digital filter (bits 2-4)
6	+64	Swap the QEA/QEB bits in terms of encoder use. Effectively reverses the direction sense for the counter.
7	+128	SI2/SO2 "Daisy-chain" TTL-serial enable. If set, SI2/SO2 are used as daisy-chain serial. If clear, those pins are available as generic TTL I/O lines

8-9		Configure CNT0 special options. CNT0 may be attached to ENCODER count events through these bits.
		9 8

	+0	0 0 CNT0 is not connected to the encoder
	+256	0 1 Always generate a PULSE request on each MAXCNT event
	+512	1 0 Wait for IO0 going low before enabling PULSE requests on MAXCNT events
	+768	1 1 Wait for IO0 going high before enabling PULSE requests on MAXCNT events
10-11		Configure CNT1 special options. CNT1 may be attached to IO1 edge events through these bits.
		9 8

	+0	0 0 CNT1 is not connected to IO1
	+1024	0 1 <Reserved: at present, not used>
	+2048	1 0 Wait for IO1 going low before enabling PULSE sequence generation
	+3072	1 1 Wait for IO0 going high before enabling PULSE sequence generation
12-13		Configure CNT2 special options. CNT2 may be attached to IDX edge events through these bits.
		9 8

	+0	0 0 CNT1 is not connected to IDX
	+4096	0 1 Generate one pulse on each IDX event
	+8192	1 0 Wait for IDX going low before enabling PULSE sequence generation
	+12288	1 1 Wait for IDX going high before enabling PULSE sequence generation

xU – Set selected TTL lines low

This command allows you to selectively set a subset of the IO bits 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	PWM1L
1	+2	PWM1H
2	+4	PWM2L
3	+8	PWM2H
4	+16	PWM3L
5	+32	PWM3H
6	+64	<no connect>
7	+128	<no connect>
8	+256	IO0
9	+512	IO1
10	+1024	IO2
11	+2048	IO3
12	+4096	<no connect>
13	+8192	SO2
14	+16384	SI2
15	+32768	<no connect>
16	+65536	RB0
17	+131072	RB1
18	+262144	<RB2, but not available for output>
19	+524288	<IDX, but not available for output>
20	+1048576	<QEA, but not available for output>
21	+2097152	<QEB, but not available for output>
22	+4194304	RB6
23	+8388608	RB7

For example, to set IO2 and PWM1H 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 'd' command.

xV – Verbose mode command synchronization and processor clock control

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. It also includes the capability of resetting the processor clock rate. **By default (after power on and after any reset action), the board is normally 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. It is also configured to operate the processor clock at full speed (about 29.5 MHz).** 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. 'V' is ignored if you fail to provide a parameter (i.e., 'V' alone results in the board ignoring the command, while '0V' causes the normal 'V' operation with a parameter of '0').

The firmware actually recognizes and responds each new command within a few microseconds of receipt of the stop bit of the received character. In most designs, this will not be a problem; however, since all commands issue an '*' upon completion, and they can also (by default) issue a <CR><LF> pair before starting, it is quite possible to start receiving data pertaining to the command before slower non-interrupt based devices can change their state! In microprocessor, non-buffering designs (such as with the Parallax, Inc.^{tm} Basic Stamp ^{tm} series of boards), this can be a significant issue. The firmware handles this via a configurable option in the 'V' command. If enabled, the code will delay 1 millisecond upon receipt of a new command character. This really means that the first data bit of a response to a command will not occur until at least 9 bit intervals after completion of transmission of the stop bit of that command (about 900 uSeconds at 9600 baud); for the Basic Stamp^{tm} this is quite sufficient for it to switch from send mode to receive mode.

The verbose command is bit-encoded as follows:

Bit	SumValue	Use When Set
0	+1	Send <CR><LF> at start of processing a new command
1	+2	Delay about 1 millisecond before transmission of first character of any command response.
2	+4	Send numeric responses in hexadecimal instead of decimal.
3	+8	Used in conjuction with Bit 4 to reset the processor clock speed. If Bit 4 is set, then bit 3 defines the clock rate: +0: Use standard high speed clock (29.5 MHz) +8: Use low speed clock (7.4 MHz)
4	+16	If set, then use bit 3 to redefine the processor clock rate. If not set, then the clock rate is left unchanged, and bit 3 is ignored.
5	+32	If set, code always sends responses: incoming characters do NOT stop pending data. WARNING -- This can cause loss of incoming commands, if 4 or more characters sent while the 64 character transmit buffer is full!

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

For example,

```
0V
```

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

```
3V
```

would enable transmission of the <CR><LF>sequence, preceded by a 1-character delay.

The complete table of options when not changing the processor clock rate is:

Value	Delay First	<CR><LF>	Hex mode reports
0	No	No	No
1	No	Yes	No
2	Yes	No	No
3	Yes	Yes	No
4	No	No	Yes
5	No	Yes	Yes
6	Yes	No	Yes
7	Yes	Yes	Yes

Changing the processor clock rate using the 'V' command

The 'V' command may also be used to change the processor clock rate for the board. The firmware currently supports two rates: 29.5 MHz (the default) and 7.4 MHz (low power). When the board is fully idle, with the encoder disabled and no TTL lines configured as outputs, then operating at the reduced clock rate changes the board draw from its standard 150 mA down to 90 mA. This is quite useful to keep the processor temperature down, and to reduce current draw during idle time on battery applications.

The side effects of changing the processor clock rates to the lower clock speed can be summarized as:

1. Pulse count rates (i.e., 'W', 'w' and 'g') have units of 20 microseconds instead of 5 microseconds.
2. The board maximum encoder edge detector rate reduces to 2 Mhz instead of 8 MHz (thus the maximum pulse rate on QEA or QEB is about 500 KHz).
3. If you currently have any of the standard rates set to a value above 14,400, you will need to re-issue the matching command to reset that rate to a legal value. Otherwise, the board response will be erratic.

Warning! It is an absolute requirement that your code wait for the '*' response whenever a 'V' command is given that resets the clock rate! The firmware has to reset the baud rate of the serial clock to match the new processor rate. This action will garble any pending incoming data, causing either incorrect command execution (since the data may become almost anything), or missed command execution.

To change the clock rate, you add one of two values to your standard 'V' command (from the above table):

+16:	Set to the normal high speed clock (29.5 MHz)
+24:	Set to the low speed clock (7.4 MHz)

For example, to set to the low speed clock with the common <CR><LF> responses, issue the command:

```
25V
```

To set the high speed clock with reduced response and the 1 millisecond delay, send:

```
18V
```

Remember to wait for the '*' before you send any more serial data to the board!

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 5 microseconds each. This means that 1 millisecond may be specified by the value of 200: the maximum allowed count is 65535, which is 327.675 milliseconds.

If the pulse generator is not yet running, '200W' acts identically to:

200W	Set first-to-second edge time to 1 millisecond
200w	Set second-to-first edge time to 1 millisecond
1x	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 30 microseconds high, 1 microseconds low (then the next pulse would start), issue the sequence:

6W	30 microsecond high
1w	5 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.

xx – Set count to first pulse generator edge for current generator(s)

The 'x' command may be used to force a different start-time to the first edge of the first edge of the first pulse (will be active when the associated generator gets its next 'G' command).

Normally, the 'b' command defaults to a value of '1'. You would use differing values as a technique to allow two pulse generators to operate in differing phases.

As with all of the pulse commands, the units are in terms of 5 microsecond clock cycles.

x= – Assign current encoder value

If the appropriate mode is enabled via the 't' command, then the standard encoder 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 32 bit encoder counter. The '=' command allows that counter to be reset to user specified values.

Please see the general value report command '?' (starting on page 44), for information on how to retrieve encoder counter data.

For example,

```
0=
```

will assign the encoder counter 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 (or memorized) defaults.

The reset command also selects the following settings, unless they have been overridden by the '123456789e' command:

- **7M** – Select all three pulse generators for actions
- **0G** – Stop all pulse generators
- **0**= – Define encoder to be at location 0
- **0d –** All I/O lines are inputs
- **0D –** All programmable outputs are set to 0.
- **1V** – Set <CR><LF> sent at start of new command, no transmission delay time
- **9600b** – Set communications baud rate to 9600 baud
- **65535M** – Set MAXCNT to maximum legal value
- **100m** – Set encoder minimum delta to 100 counts
- **128t** – SI2/SO2 are set as TTL-Serial ports, encoder is off
- **1W** – Pulse widths are 5 microseconds high, 5 microseconds low
- **1x** – Start pulse delay (first pulse only) is 5 microseconds

? – Report status

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

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

1-6: Report TTL inputs

Useful locations:

- 1: Current PWM inputs
- 2: 'IO0' through 'IO3' inputs
- 3: SO2 and SI2
- 4: RB0-7, including the encoder inputs
- 5: all inputs, encoded identically to the TTL output commands, but with bits 24 to 27 added as the counter status summary (see report 6)
- 6: Counter run status summary

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

-1 to -18: Do selected one of the following reports

- -1; Report current encoder location
- -2; Report current counter flags
- -3; Report current counter run mode
- -4; Report PWMx bits controlled by current encoder
- -5; Report pulse delay
- -6; Report edge 1 count
- -7; Report edge 2 count
- -8; Report count of clocks left for current pulse edge
- -9; Report count of pulses left if stream of pulses being sent
- -10; Report total count of pulses sent since last -10? report
- -11; Report MAXCNT value
- -12; Report current software version and copyright
- -13; Report IO Configuration value ('t' command)
- -14: Report IO port directions ('d' command)
- -15: Report current 'Trigger' mask value ('S' command)
- -16: Report the encoder location that triggered the FIRST encoder-triggered pulse after a 'G' command
- -17: Report the minimum encoder count ('m' command)
- -18: Report the current count of encoder interrupt cycles that are in the 'reverse' direction from that required for encoder-generated pulses

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 "M" character is sent, to signify that the report is from the FENCMCT board.
3. A single digit ('0' to '2') is sent, to indicate which counter is being reported.
4. A comma is sent.
5. The report number is sent (such as –4, for encoder 0 value).
6. Another comma is sent.
7. The requested value is reported.
8. If Verbose Mode is on, then a <crlf> is sent
9. 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.

1?: Current PWM inputs

This reports the current values of the PWM I/O ports:

Bit	***Value***	***Signal***
0	+1	PWM1L
1	+2	PWM1H
2	+4	PWM2L
3	+8	PWM2H
4	+16	PWM3L
5	+32	PWM3H

2?: 'IO0' through 'IO3' inputs

This reports the current values of the IO0 to IO3 I/O ports:

Bit	***Value***	***Signal***
0	+1	IO0
1	+2	IO1
2	+4	IO2
3	+8	IO3

3?: SO2 and SI2

This reports the current values of the SI2 and SO2 I/O ports:

Bit	***Value***	***Signal***
0	+1	<no connect>
1	+2	SO2
2	+4	SI2
3	+8	<no connect>

4?: RB0-7, including the encoder inputs

This reports the current values of the RB0-RB7 encoder input I/O port:

Bit	***Value***	***Signal***
0	+1	RB0
1	+2	RB1
2	+4	RB2
3	+8	IDX
4	+16	QEA
5	+32	QEB
6	+64	RB6
7	+128	RB7

5?: all inputs plus counter status summary (see report 6)

This reports the current values all I/O ports plus the pulse generator status summary:

Bit	***Value***	***Signal***
0	+1	PWM1L
1	+2	PWM1H
2	+4	PWM2L
3	+8	PWM2H
4	+16	PWM3L
5	+32	PWM3H
6	+64	<no connect>
7	+128	<no connect>
8	+256	IO0
9	+512	IO1
10	+1024	IO2
11	+2048	IO3
12	+4096	<no connect>
13	+8192	SO2
14	+16384	SI2
15	+32768	<no connect>
16	+65536	RB0
17	+131072	RB1
18	+262144	RB2
19	+524288	IDX
20	+1048576	QEA
21	+2097152	QEB
22	+4194304	RB6
23	+8388608	RB7
24	+16777216	CNT0 busy bit
25	+33554432	CNT1 busy bit
26	+67108864	CNT2 busy bit

6?: Counter run status summary

This reports the current values of the pulse generator status summary:

Bit	***Value***	***Signal***
0	+1	CNT0 busy bit
1	+2	CNT1 busy bit
2	+4	CNT2 busy bit

0?: Report standard reportable 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:

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

Where:

- M identifies the report as FENCMCT generated
- 0 is the report number; 0 is the 'all' report
- a is the value for the current encoder reading (report "-1")
- b is the value for the current counter flags (report "-2")
- c is the value for the current counter run mode (report "-3")
- d is the value for the PWMx bits controlled by current encoder (report "-4")
- e is the value for the pulse delay (report "-5")
- f is the value for the edge 1 count (report "-6")
- g is the value for the edge 2 count (report "-7")
- h is the value for the count of clocks left for current pulse edge (report "-8")
- i is the value for the count of pulses left if stream of pulses being sent (report "-9")
- j is the value for the total count of pulses sent since last -10? report (report "-10")
- k is the value for the MAXCNT value (report "-11")

-1?: Report current encoder location

This reports the current encoder location.

For example,

```
        -1?
        M0,-1,170
        *
```

would mean that the encoder is reporting that it is at location 170.

-2?: Report current current counter flags

This reports the control flags associated with the current counter. The format of this value may change between firmware versions, in terms of new bits being added. However, the following bits are defined for all firmware versions described by this manual:

Bit	***Value***	***Signal***
0	+1	1 means pulse sequence is active, 0 means idle
1	+2	0 means counting edge 1, 1 means edge 2
2	+4	If set, first pulse has not yet occurred (startup mode)
3	+8	If set, pulse generation is legal (new pulses may be generated)
4	+16	If set, pulse generation sequence is armed (ready to start pulses based on a TTL input event)

-3?: Report current counter run mode

This reports the current counter run mode selections.

The actual values here will change version to version of the firmware.

-4?: Report bits controlled by current CNT generator

This reports the current PWMx bits controlled by the current CNT generator.

-5?: Report pulse delay

This reports the requested delay before starting the first pulse of a series.

-6?: Report Edge 1 count

This reports the edge 1 count.

-7?: Report edge 2 count

This reports the edge 2 count.

-8?: Report count of clocks left for the current pulse edge

This reports the count of clocks left for the current pulse edge.

-9?: Report count of pulses left if stream of pulses being sent

This reports the count of pulses left if stream of pulses being sent.

-10?: Report total count of pulses sent since last -10? report

This reports the total count of pulses sent since last -10? report.

It then resets that count to 0.

-11?: Report MAXCNT value

This reports the MAXCNT value.

-12?: Report current software version and copyright

This reports the software version and copyright.

For example,

```
-12?
```

could report:

```
FENCMCT 5.01 January 16, 2014
Copyright 2014 by Peter Norberg Consulting, Inc.  All Rights
Reserved.
*
```

-13?; Report IO Configuration value ('t' command)

This reports the current 't' command configuration value.

-14?: Report IO port directions ('d' command)

This reports the current I/O port directions ('d' command)

-15?: Report current 'Trigger' mask value ('S' command)

This reports the current value used as a trigger mask, as defined by the 'S' command.

-16?: Report encoder location associated with the first pulse generated by an encoder event

This reports the encoder location associated with the first pulse generated after a 'G' command on counter 0, if that counter is attached to pulses generated from encoder MAXCNT events.

-17?: Report the initial encoder offset to use on '=' command (as defined by the 'm' command)

This reports the value set by the 'm' command, which is the offset that will be loaded into the encoder whenever an '=' command is executed to reset the encoder location.

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 5D
```

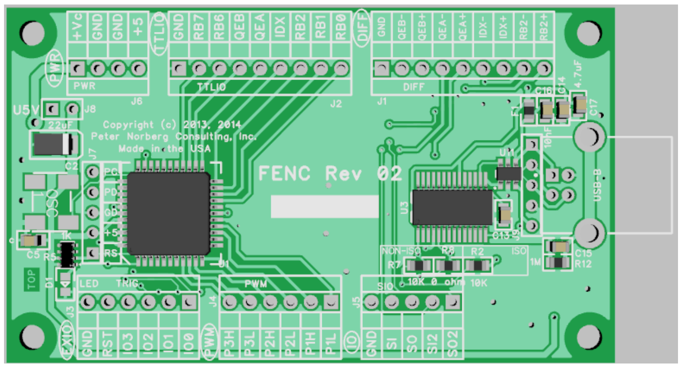
would actually request a "Set digital outputs to 5". Since the " " command is illegal, it is ignored; however, it terminates interpretation of the number which had been started as 12.

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

An example of the FENC board is shown below.



Board Size

The board, oriented as shown on this page, is 3 inches wide by 1.75 inches high.

Mounting Requirements

The FENC 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 1.5 inches apart. Thus, when the board is positioned as shown above, their positions are:

(0.125, 0.125), (2.875, 0.125),
(0.125, 1.625), (2.875, 1.625)

Connector Signal Pinouts

There are eight connectors on each board.

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

- Debugger connector (5 pin SIP header in middle of board)
- EXIO – Board reset (RST) and trigger-based IO ports 0 to 3.
- PWM – 6 TTL output signals (can be reset to be inputs)
- IO – TTL-Serial I/O signals

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

- PWR – Power and connector (upper left)
- TTLIO – Generic TTL I/O connector for encoder and extended TTL I/O
- DIFF – Differential inputs for quadrature encoder
- USB Serial connector (center right hand side)

Debugger connector (J7)

Pin	Name	Description
1	RS	Board reset
2	+5V	+5 for debugger
3	GD	Ground for debugger
4	PD	Programming data
5	PC	Programming clock

Normally, you will not use this connector. There is one special feature available from it, however – it can be used to generate a full board factory reset, if a jumper is installed between the PD and PC pins and the board is power cycled.

Factory Reset via shorting PD to PC

If you install a standard 0.1" jumper header on pins 4 and 5 (PD to PC) of the debugger connector and then apply power to the board for a few seconds, the board will reconfigure itself to its factory default settings. This is exactly equivalent to issuing the "987654321e" command, followed by a reset ("!") command. This allows you to force the board to a known state if you have really confused the system with improper settings.

EXIO – Board reset (RST) and trigger-based IO ports 0 to 3

Name	Description
GND	Ground reference for all signals
RST	Board reset: ground pin to reset board
IO3	Generic TTL I/O signal IO3
IO2	Generic TTL I/O signal IO2
IO1	Generic TTL I/O signal IO1
IO0	Generic TTL I/O signal IO0

This connector gives access to the board reset, as well as the 4 event trigger IO signals IO0 to IO3.

PWM – 6 TTL output signals (can be reset to be inputs)

Name	Use
P3H	PWM3H
P3L	PWM3L
P2H	PWM2H
P2L	PWM2L
P1H	PWM1H
P1L	PWM1L

The PWM connector is normally used for output of CNT based pulses.

IO – TTL-Serial I/O signals

Pin	Name	Description
1	GND	Board ground reference
2	SI	TTL-Serial input for board
3	SO	TTL-Serial output from board
4	SI2	Daisy-chain TTL-serial input to board
5	SO2	Daisy-chain TTL-serial output from board

The IO connector is normally used for TTL-Serial I/O operations.

The SI and SO pins are the TTL serial versions of the USB connection. If you attach your own TTL-serial signals to these lines, then USB communications get preempted by your signals.

The SI2 and SO2 pins are normally used for daisy-chain communications, allowing multiple boards to be connected to one USB line. They may be reprogrammed by you to be generic TTL I/O lines, if needed.

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)

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. The first two methods involve your providing a power supply and your making certain that the U5V jumper is NOT installed. The third method allows you to tap the USB system for power.

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.*
3. You may elect to use the USB connection as your power source. In this case, you connect nothing to the PWR connector, and you insert a jumper in the U5V header, enabling tapping of power from the USB system. **NEVER** connect power to the PWR connector if you have the U5V jumper installed.

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!***

TTLIO – Generic TTL I/O connector for encoder and extended TTL I/O

Name	**Description**
GND	Ground reference
RB7	Generic TTL I/O 7
RB6	Generic TTL I/O 6
QEB	Generic TTL input 5, or quadrature input A
QEA	Generic TTL input 4, or quadrature input B
IDX	Generic TTL input 3, or quadrature INDEX input
RB2	Generic TTL input 2
RB1	Generic TTL I/O 1
RB0	Generic TTL I/O 0

This connector gives you access to the TTL versions of the quadrature encoder signals (QEA, QEB and IDX) as well as generic I/O signals for general use.

Four of the signals on this connector (QEB, QEA, IDX and RB2) are always configured as inputs, and are cross-connected to the DIFF connector which has differential versions of these same signals. Only connect your signal source to one or the other of the connectors (i.e., use QEA+/QEA- from the DIFF connector or use QEA from this connector, but not both).

The remaining signals (RB7, RB6, RB1 and RB0) are available for generic use by your application. They are controlled through use of the various TTL I/O commands, and they also support (rather imprecise) analog input for 0 to 5 volt analog signals.

Differential inputs for quadrature encoder

Pin	**Name**
1	GND
2	QEB-
3	QEB+
4	QEA-
5	QEA+
6	IDX-
7	IDX+
8	RB2-
GND	RB2+

This connector gives access to the differential-input versions of the encoder system. This allows you to use encoders that have differential outputs as opposed to TTL outputs. The signals get buffered through a standard differential receiver, and then are routed through a 10K resistor to their matching signals on the TTLIO connector. This means that you can either connect your signal to the differential input (DIFF) or the TTLIO version, but not both. The TTLIO version would take precidence.

USB connector

This provides you with the USB communications for the system.