User Guide

Instructions for EZ4AXIS17XR-ST

Preliminary Release

Manual revision 1.15
Sep, 2017

Important Notices

Life and Safety Policy

Without written authorization from the AllMotion company President, AllMotion, Inc. products are not authorized for use as critical components in life support systems, for surgical implant into the body, or other applications intended to support or sustain life or any other applications whereby a failure of the AllMotion, Inc. product could create a situation where personal injury, death or damage to persons, systems, data or business may occur.

AllMotion, Inc.
30097 Ahern Avenue
Union City, CA 94587
USA

Tel: 408.460.1345
Fax: 408.358.4781

Technical Support: 408.460.1345
Sales: 510.471.4000

Website: www.allmotion.com

Copyright © 2017, AllMotion, Inc. All rights reserved.

The following are trademarks of AllMotion, Inc.: AllMotion®, EZStepper ®, EZServo®, EZBLDC™, EasyBLDC™. Other names, brands, and trademarks are the property of others.

AllMotion, Inc. assumes no responsibility or liability for information contained in this document. AllMotion, Inc. reserves the right to make corrections, modifications, enhancements, improvements, and other changes to its products, specifications, and services at any time and to discontinue any product or service without notice. The information contained herein is believed to be accurate and reliable at the time of printing.

Table of Contents – note: page numbers approximate

1. Quick guide

This EZ4AXIS17XR can be commanded by sending simple text messages to the drive. It is not necessary to compile software etc. For example sending the text ***P1000*** makes the drive go in the Positive direction **1000** steps. The drive will also respond back with a text message. The text messages can also be stored on the drive and executed on power-up. Text messages can be sent via RS232, RS485, USB or CANBUS.

Quick setup

Hook up the EZ4AXIS17XR using the Quick Start guide and the wiring diagram from links at the bottom of appropriate web page:
http://www.allmotion.com/Stepper_Pages/EZ4AXIS17XRdescription.html

Per quick start guide find the com port that appears *and disappears* when the USB is plugged in and out. This is the com port for the drive. If a *new* com port does not appear, then install the windows/mac drivers manually.

CAUTION: Do not unplug the motors when the power is on, because the inductance of the motor will generate a high voltage when the current in the motor is interrupted, which will damage the drive.

1. Using the EZCommander windows App, issue the command */1&*. The drive should respond with its firmware version.
2. With the power off plug in Motor #1. Turn on power, issue the command */1P1000R*. The drive should move forward 1000 microsteps.
3. Turn off power and plug in up to four motors.
4. Issue the command

   /1aM2P1000R. Motor 2 should move.

   /1aM3P1000R Motor 3 should move.
5. Issue the command

   /1P1000,1000,1000,1000R

   All 4 motors should spin
6. Issue the command

   /1P1000,1000,,1000R

   Motors 1, 2, and 4 will spin. *(note comma positions)*

   /1V100,200,300,400R sets motor speeds
7. */1?aA* reads back the position of all four motors.

Please see the additional examples “Single-Axis programming supplemental examples” starting on page 39, “Multiple-axis/multiple drive supplemental examples” on page 41, and the “Command list” on page 71.

2. Introduction

About this manual

This document describes how to program and operate the EZStepper® EZ4AXIS17XR (Stepper Controller + Driver).

These sections are included:

- **Basic Programming Operations**. (Starting page 14) This describes how to set up basic motor moves, first on one axis and then multiple axes.
- **Advanced Programming Operations.** (Starting page 28) This describes how to set up more complex moves, including conditional moves based on signals at the inputs. Many examples are provided. This section also contains essential information about how the product operates, and describes operations not covered in the Basic Operations section.
- **Sections focused on specific Programming applications.** (Starting page 43) Limits, Digital IO (switch-controlled) Applications, Analog Control Applications, Homing, and Using Encoders.
- **Appendices.** (Starting page 68) The appendices contain reference information to supplement the instructions, such as an extensive command set for the product. There is also additional technical information that you may find helpful in applying the product.

Before you start

Obtain communications software

- If you are using a Windows-based system to communicate with your AllMotion product, we recommend that you download and install the EZ Commander™ application from the AllMotion website Support page.

  Alternatively, a terminal program such as HyperTerminal may be used.
- If you are using a Macintosh system to communicate with your product, any text-based terminal program will be adequate.

  NOTE: The EZStepper® communicates over the RS485 EZ bus at 9600 baud, 1 stop bit, no parity, no flow control. If necessary, this can be changed with the *b* command, after getting hooked up. This is covered in "Essential setup information" on page 28.

Obtain needed documentation

Obtain the following for the EZ4AXIS via links at the bottom of this web page: http://www.allmotion.com/Stepper_Pages/EZ4AXISdescription.htm http://www.allmotion.com/Stepper_Pages/EZ4AXIS17XRdescription.html

- EZ Start document (EZ Starter Kit Instructions) for your product. This will get you started with setting up communications and confirming operation.
- Wiring diagram. This will serve as a guide to hookups and various ways of applying your AllMotion product.
- Data sheet.

Gather needed equipment

Ensure that you have compatible stepper motor(s), a PC or other device capable of running a terminal program, and a power supply. Typically use a 12V or 24V capable of 3A. If starting out use a current limited lab supply set to 0.5A current limit.

- Do not connect or disconnect motor cable while power is on. This causes a high voltage spark due to the inductance of the motor, and can damage the chips on the drive. This includes loose connections.

About the EZStepper® command language

Features

This EZ4AXIS can be commanded by sending simple text messages to the drive. It is not necessary to compile software etc. For example sending the text ***P1000*** makes the drive go in the **P**ositive direction **1000** steps. The drive will also respond back with a text message. The text messages can also be stored on the drive and executed on power-up. The drive can also test the status of inputs and execute commands conditional on the status of an input with no computer attached.

Syntax

The text message commands consist of single alpha characters usually followed by a numeric value. The alpha character represents "what to do" and the numeric value represents "how much to do it." A complete command string contains a start character, a device address, a command, and a Run command—which tells a particular motor (or axis) to accept the command string.

Example: */1P1000R*

Text message command string breakdown:

/ Start character. Tells the EZStepper® that a command is coming in.

1 Device address. This is the number set on the rotary mechanical Address switch on the Drive. .

P1000 Command. Move 1000 steps in the positive direction. From wherever the motor currently is. By default on power up this command goes to motor#1.

R Run the command. This is the end of the command string.

Multiple commands can be entered in a single string and they are executed one at a time starting with the command on the left.

Example: */1V1000P1000V2000P1000R*

where the first *P1000* will happen at a velocity of 1000 and the second *P1000* will happen at a velocity of 2000.

3. Basic programming operations

This section describes how to enter basic commands to the EZ4AXIS. A complete listing of commands is contained in Table 1 on page 71.

First things

Make sure system is connected and operating

Follow the instructions in the EZ Start document for EZ4AXIS17XR, to make sure you have it hooked up properly and it is communicating with your host PC. The EZ4AXIS17XR board should be set to address 1 as described in the EZ Start document.

TBD Add link to easy start here

Basic move commands to Axis 1

These are the basic move commands:

P Move motor to relative position in positive direction (microsteps).

D Move motor to relative position in negative direction (microsteps).

A Move motor to absolute position in microsteps.

Z Move motor to home position.

1. Move Axis 1 motor to relative position in positive direction (the *P* command).

NOTE: *Relative* position refers to movement measured from the current position of the motor.

Send this command string: */1aM1P1000R*

Result: The motor on Axis 1 rotates 1000 microsteps in the positive direction from its current position. It then stops and holds its new position until another command is received.

Command string breakdown:

/1 Start character; select address 1 on the EZ bus.

aM1 Select Axis 1 on the board.

P1000 Move 1000 microsteps in positive direction from current position.

R Run the command.

2. Move Axis 1 motor to relative position in negative direction (the *D* command).

Relative position refers to movement measured from the current position of the motor.

Command string breakdown:

/1	Start character; select address 1 on the EZ bus.
aM1	Select Axis 1 on the board.
D1000	Move 1000 microsteps in negative direction from current position.
R	Run the command.

3. Move Axis 1 motor to absolute position (the *A* command)

On power up the Absolute Position is set to zero. The absolute position will also be set to zero when using the Home (*Z*) command at the point when the Home is triggered. The absolute position is a measure of the total distance moved from the point at which the absolute position is set to zero.

Send this command string: */1aM1A1000R*

Result: The motor on Axis 1 rotates to absolute position 1000 microsteps. It then stops and holds its new position until another command is received.

Command string breakdown:

/1	Start character; select address 1 on the EZ bus.
aM1	Select Axis 1 on the board.
A1000	Move (rotate) to absolute position 1000.
R	Run the command.

Note that issuing this command again will not make the motor move because it is already at absolute position 1000. The current absolute position can be read back by issuing */1?0*.

4. Home the Axis 1 motor (the *Z* command)

This command causes the motor to turn in the negative direction until the home sensor is tripped or, once the sensor is tripped, the position is set to zero. This allows the drive to initialize a mechanism to a known position. Details are located in "Homing to opto/switch (default *N1* mode)" beginning on page 51.

Example: Set up a normally closed switch, or opto between the home input and the ground input on the ten-pin Axis 1 Home connector.

Send this command string and push the switch while it's running: */1aM1Z10000R*.

Result: The motor on Axis 1 Drive 1 rotates to the home position until the switch changes state, and then the position for that axis is set to zero.

Command string breakdown:

/1	Start character; select address 1 on the EZ bus.
aM1	Select Axis 1 on the board.
Z	Move to home position.
10000	Maximum number of microsteps or encoder counts that the motor is allowed to move toward home in search of the home flag before the program declares an error.
R	Run the command string.

NOTE: From this point onward, command string breakdowns will not always be shown.

Send commands to other axes

To send any command to Axes 2, 3, or 4, simply change the *aM1* in the command string to one of the following:

aM2	Axis 2
aM3	Axis 3
aM4	Axis 4

Example:

*/1**aM2**A1000R* goes to Axis 2.

*/1**aM3**A1000R* goes to Axis 3.

Send commands to multiple axes (multi-axis commands)

Overview

Some commands can be issued to one, some, or all four axes on the drive simultaneously. Thus they are "multi-axis" commands. These are the multi-axis commands:

L	Set acceleration (sets same value all axes).
V	Set velocity (sets same value all axes).
h	Set hold current (sets same value all axes).
m	Set move current (sets same value all axes).
P	Move forward to relative position.
D	Move backward to relative position.
A	Move to absolute position.

All other commands require individually selecting the axis and issuing the command. */1aM2P1000R* etc.

When issuing a multi-axis command, the axis positions are delimited by commas, beginning with Axis 1 on the left and ending with Axis 4 on the right, as shown in the following examples.

Send a move command to all four axes

Send this command string: */1P1000,1000,1000,1000R*

Result: All four axes move 1000 microsteps in the positive direction.

You may send different values to different axes. For example: */1P1000,300,1000,300R* moves Axis 1 and 3 1000 microsteps and Axes 2 and 4 300 microsteps.

NOTE:

- Negative numbers would designate negative movement for the otherwise positive-direction *P* command. If the *D* command (negative move) were used, negative numbers would designate positive movement. Example: */1P1000,-500,1000,-500R* would move Axes 1 and 3 1000 microsteps positive, and Axes 2 and 4 500 microsteps negative.
- When several multi-axis commands are placed in a string one after the other, by default each motor will wait until all have completed their motion before responding to the next command. I.e., they all go to the "4-axis coordinate" given by each command prior to starting the next command. Example of such a string: */1A1000,200,300,10A200,500,200,900A200,300,500,78R*.

  See "Motor sequencing in multi-axis commands" on page 30.
- When a multi axis command is issued, it will reset the default single axis command destination to Motor 1. (Inherent *aM1* command)
- The target position can be changed using an "immediate" command even while a move is being executed. (See "Making on-the-fly parameter changes" on page 31.)

Send a move command to just two axes

Omit values for the non-addressed axes, but retain commas.

Send this command string: */1P1000,,1000,R*

Result: Axes 1 and 3 move 1000 microsteps in the positive direction. Axes 2 and 4 do not move. The commas marking the positions of Axes 2 and 4 are retained. Note that motors can be started at different times by issuing commands as "*immediate commands*" while some motors are moving. An "*immediate command*" is a single command issued while another command is already executing. And will interrupt/add to the currently executing command.

Set velocity, acceleration, and current

The following commands control velocity, acceleration, and motor current.

V Maximum velocity (speed): This sets the maximum velocity for a move, which is reached after an acceleration period set by the *L* command.

Single Axis Example: */1aM1V1000R*

Multi Axis Example: */1V100,200,200,300R*

Or /1V100,,100,R to affect motors 1 and 3only

L Acceleration (acceleration factor): This sets the acceleration factor, which controls the amount of time required for the motor to reach the maximum velocity (acceleration ramps). The EZAXIS1 equation for acceleration is: Acceleration (in microsteps / sec^2) = (L value) x TBD For example, if V=10000 and L=1, it will require TBD seconds to reach final velocity.

Single Axis Example: */1aM1L100R*

Multi Axis Example: */1L100,200,200,300R*

Or /1L100,,100,R to affect motors 1 and 3only

m Maximum move current: This sets the current level when the motor is moving. More current provides more force and more acceleration. Move current is given as a percentage of the maximum output drive current for the device (m100 = 0.5A for EZ4AXIS17XR).

Single Axis Example: */1aM4m30R*

Multi Axis Example: */1m10,20,50,30R*

h Hold current: This sets the current when the motor is stationary. The hold current prevents the motor from slipping against any disturbance when not moving. Hold current is expressed as a percentage of the maximum output drive current for the device (h50 = 0.25A for EZ4AXIS). (50% is max allowed for hold current)

Single Axis Example: */1aM3h20R*

Multi Axis Example: */1h10,20,20,30R*

Store and recall programs

Overview

Command strings for later recall can be stored in the EEPROM on the EZ4AXIS board.

1. Store a command string (the *s* command)

Store specific command string in designated EEPROM location. the s command is followed by a number ranging from 0 to 63, indicating an EEPROM location.

Example: */1s2P1000R*

Result: The command *P10000* is stored in EEPROM location 2 as specified by the *s2* command, and may be referred to as program 2.

2. Recall (run) a stored command string (the *e* command)

Run the command string stored in designated EEPROM location.

Example: */1e2R*

Result: Executes (recalls) the command string stored in EEPROM location 2.

3. Erase a specific EEPROM location

To erase a location, use the store command without any commands indicated.

Example: */1s2R*

Result: EEPROM location 2 is erased.

4. Erase a stored command string.

To erase a location, use the store command without any commands indicated.

Example: */1s2R*

Result: EEPROM location 2 is erased.

5. Automatically run a stored program on power-up

The command string in location *0* is always executed on power-up. If we used *0* instead of *2* in the above example, this program would execute automatically on power-up.

Example: */1s0A0P10000R*

Result: The command *A0P10000* is stored in EEPROM location 0. Cycling the power will cause *A0P10000* to be executed and the drive to go forward 10000 steps on power-up.

NOTES

- Programs cannot be running while programming the EEPROM. Terminate any strings or loops with */1T* prior to issuing a store command.
- Program storage takes approximately one second to execute. The drive will not respond during this time.
- Each of the 0-63 memory location can accept up to 25 full commands per string, or 256 characters, whichever is less.

  Example: */1s0A0A100gP1000m1000G5e1R*
- Excessive characters will overwrite the 256th character repeatedly until the R (run) command, which is not overwritten because it is the final character in the string.
- Programs can call each other.

  Example:

  /1s0A0e1

  /1s1A1000e0R

  Will cause the motor to go between *A0* and *A1000* on power-up.
- Single axis commands will go to whichever motor was selected with the last */1aM2R* etc motor selection command. This command can be stored in the EEPROM to ensure the correct motor moves. */1s0A0aM2P1000R* will move motor #2 1000 steps on power-up.

  NOTE: There is no relationship between the "Send String" numbers in the EZCommander Windows application and the storage locations in EEPROM specified by the *s* command.

Reading a stored program from EEPROM

To read the contents of a memory location, first execute the command in that memory location, then read the currently running or most recent command.

Example:

1. Issue */1s13aM1A1000A0R* to store command in location 13.
2. Issue */1e13R* to execute the command in location 13.
3. Issue */1$* to read the currently-running or most recent command string.

   Example result: *aM1A1000A0 No Error*.

NOTES

- Do not include the execute query commands in the same command string as an action command. Eg /1A1000$R won't work.
- The *$* command may not be able to read very long command strings. On older firmware, attempting to read very long command strings may kill board operation. If this happens, power cycle the board.

When a string length exceeds memory location capacity

Each EEPROM location has a capacity of 256 characters, including the run (*R*) command. If the command string exceeds this number, it can be broken into two smaller strings that reside in two different memory locations. At the end of the string that runs first, a command to execute the contents of the second memory location is inserted just before the *R* command. If, for example, the second string is in location 7, *e7* (execute the contents of EEPROM location 7) would be inserted.

NOTE: When a command string exceeds 256 characters, the last character is repeatedly overwritten by the next character in the string until, finally, the R (run) character is written. Thus, such a string will always attempt to execute.

Create loops

One or more commands can be looped, by placing a lower case *g* at the beginning of the and an upper case *G* at the end of the command(s), a number after the upper case G sets the number of times the loop is to be repeated.

The following examples describe how to make two basic types of loops.

NOTE: The example also introduces the *M*, or wait, command.

1. Create a loop that repeats a specific number of times

Example: */1aM2gP1000M500D1000M500G10R*

Command breakdown:

/	Start character. Tells the EZSteppers® that a command is coming in.
1	Device address, (set on address switch on device).
aM2	Axis address, in this case Axis #2.
g	Start a loop.
P1000	Move 1000 microsteps in the positive direction.
M500	Wait for 500 milliseconds.
D1000	Move 1000 microsteps in the negative direction.
M500	Wait for 500 milliseconds.
G10	Repeat 10 times beginning at the location of the *g* command.
R	Run the command.

Result: The command string following the *g* command executes and ends when the *G* command has repeated 10 times.

Issue */1T* to terminate the loop at any time.

2. Create an endless loop

Change *G10* in the preceding example to *G0* (Gzero).

Result: The loop starts and continues forever unless a *T* command is issued. To terminate this loop, issue */1T*. (Do not use */1T2*, */1T3* etc. to terminate a *G* loop since the behavior is undefined and may change in the future.)

Note that if a loop is running a /1T must be issued before making any changes or sending a new command. The only exception is a single immediate command which will be inserted into the loop, one time, at the time of sending.

3. Exit a loop upon input level change method 1.

Example:

/1s1R—store nothing in program 1 in the EEPROM.

1aM1gP1000M1000S02e1GR—loop of motion and wait.

Result: With switch 2 input high the loop executes normally. When Switch 2 is brought low the program executes the *e1* command and jumps out of the loop.

4. Exit a loop upon input level change method 2.

Example:

1aM1gP1000M1000S02G0R—loop of motion and wait.

Result: With switch 2 input high the loop executes normally. When Switch 2 is brought low the program skips the *G* command and stops.

This mode also works for the following strings:

1aM1gP1000M1000S02G0A10000R

where *A10000* executed upon switch closure or even

1aM1ggP1000M1000S02G5A0GR

where the inner loop is exited upon switch closure.

5. Halt a loop pending level change.

Example: Send the following command strings.

1aM1gH02A1000A0GR

Where loop only runs when Switch 2 is low.

1aM1gH02H12A1000A0GR

Where loop runs once for every low/high transition of Switch 2.

NOTE: More complex loops are described in Section 4, “Advanced programming operations.”

6. Use limits to move smoothly until sensor cut

Send these command strings:

1gaM1n2P0n0P1500gaM2P1000aM1P1000G8aM1P25000GR

Say motor 1 is a conveyor belt that moves a microtiter plate until its under a syringe moved by motor 2. Use a sensor wired to the high limit of motor 1, the P0 moves until the microtiterplate is under the syringe then stops, no limits are turned off, the plate moves forward 1500 until plate e well is under syringe, plate is filled with aM2P1000 eight times. Plate is kicked out with P25000 and cycle repeats

Terminate a program during execution

To immediately terminate any executing command or string on the board, issue */1T*.

Note that the string is still in the command buffer, and may be executed from the beginning by issuing */1R*.

If a multi-axis command is in progress, one axis at a time may be terminated by using */1T1, /1T2, /1T3, or /1T4*.

NOTE: The *T1*, *T2*, *T3*, and *T4* commands should only be used to terminate a single multi-axis command. The behavior in a loop or multi-command string is undefined and may change.

Controlling Power Drivers (solenoid drivers)

The EZ4AXIS17XR has 16 power drivers for actuating solenoids or any device requiring a relatively large current. While each driver is capable of switching in excess of 1A the total current to the board is limited by the current capability of power connector which is 3 max.

Hookup

- Make hookups to the Power Driver Connector as shown below. There is a + and – connection for each device.

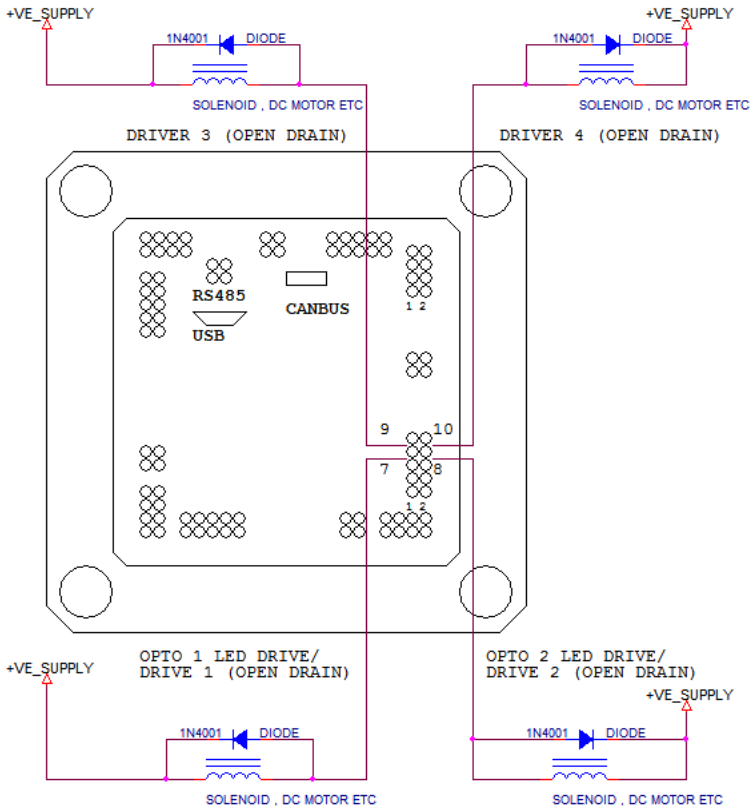


Figure 1 Power Driver Hookups

CAUTION: Do not disconnect inductive loads while power is on. The resulting spark will damage the drivers. This includes loose connections.

Programming

- To actuate a driver, issue the *J* command:

  Digits following the *J* command are interpreted as 4-bit binary equivalents: 1111 binary = 15 decimal = all drivers on

/1gJ15,15,15,15J0,0,0,0GR will toggle all outputs on and off

/1gJ1,0,0,0J0,0,0,0GR will toggle output 1 on axis 1 on and off

Note that Drive 1 and 2 have a 200 Ohm resistor to the 5V supply on the board, (for driving the optos) this will interfere with the operation of the output if a +VE supply greater than 5V is used. Please contact the factory for instructions on removing the resistor.

Bitwise On/Off (not implemented yet)

- The legacy J command described above requires shadow registers to be maintained by the user software because the command changes all outputs.

  The aJ and aaJ command address this problem by allowing bitwise set and reset of the outputs.

- The aJ command clears a particular output bit without affecting any other output bit.

  /1aJ1,0,0,0R will turn off Driver 1 on Axis1

  /1aJ0,0,0,4R will turn off Driver 3 on Axis4

- The aaJ command sets a particular output bit without affecting aany other output.

  /1aaJ1,0,0,0R will turn on Driver 1 on Axis1

  /1aaJ0,8,0,0R will turn on Driver 4 on Axis2

PWM of On/Off Ouput (not implemented yet)

- The aK command allows the outputs to be turned on and off at multiples of 10Khz intervals.

  aK Axis, Output, HiTime, LowTime Where the Times are in increments of 100uS increments.

  /1aK1,3,1,1 creates a 100uS On, 100uS Off (5Khz) PWM on axis1 channel 3

4. Advanced programming operations

Essential setup information

Motor direction of rotation considered positive

It is recommended that the positive and negative directions be set up by suitable connection of the motor wire leads. Typically if the motor does not turn in the desired direction, switch the A+ and A- leads on the motor.

Alternatively the direction can be switched by using the F command. For example */1aM2F1R* will reverse the direction considered positive for Motor 2 and */1F1,0,1,1R* will set directions for all 4 axes. This should be done once on power up only. Do not use this command to change direction during normal operation. Instead use the *P* and *D* commands. (Using this command during normal operation will result in loss of position because it instantly inverts the drive to one of the phases). Do not use this command if using encoder feedback mode. TBD Check this

Changing communication baud rate

The baud rate can be adjusted with the *b* command. Enter the command followed by the desired baud rate, which can be 9600, 19200, or from 38400 to 230400. Default baud rate is 9600.

Example: */1b19200R* for 19200 baud

Result: The baud rate of Drive 1 is set to 19200 baud.

NOTES

- The baud rate command will usually be stored as program zero and executes on power-up, so that the drive starts talking at a different baud rate. To store as program zero, add *s0 to* the command string: */1**s0**b19200R*. However do NOT store a high baud-rate command in program zero until communication has been tested at the higher baud rate.
- Contact factory for instructions on stopping the EEPROM readback on power up, if somehow communication is lost with the drive due to programming with high baud rate in program zero.
- Correct termination and strict daisy-chaining is required for reliable operation at the higher baud rates.

Readback of parameters from the drive:

Reading firmware version

Example: */1&*

This reads the firmware version on Drive 1.

Reading motor positions

To read the position of the currently selected motor, use the */1?0* command:

Example: */1aM4R* then issue */1?0* separately. This retrieves the motor position of Axis 4.

If a command has just been sent to Axis 4, only */1?0* is necessary.

/1?aA reads back the position of all four motors simultaneously.

Similarly use:

/1?V, /1?aV TBD check this works

/1?L, /1?aL TBD check this works

Reading currently running or most recent command

To read the currently-running or most recent command string, use the *$* command.

Example: Issue */1P1234P4321R*; then issue */1$* to retrieve the currently-running or most recent command.

Response will be *P1234P4321 No Error*.

Default axis selection

- If *aM* (axis selection command) is omitted in a command string, the command will go to Axis 1 by default or the last axis selected if other than Axis 1.
- If a multi-axis command has just been issued, the default for the next command will always be Axis 1.

Designating negative relative move in otherwise positive-move multi-axis command strings

Example: Use negative numbers to indicate negative relative movement.

/1P10000,-10000,10000,10000

Moves Axes 1, 3, and 4 positive 10000 microsteps, and Axis 2 negative 10000 microsteps.

Designating positive relative move in otherwise negative-move multi-axis command strings

Example: Use negative numbers to indicate positive relative movement.

/1D10000,10000,-10000,10000

Moves Axes 1, 2, and 4 negative 10000 microsteps, and Axis 3 positive 10000 microsteps.

Pre-select axis and send commands subsequently

Once an axis has been selected, subsequent single-axis commands and queries will be directed to that axis until another axis is selected or a multi-axis command is issued (multi-axis commands reset default axis to Axis 1).

Example:

/1aM4R	Selects Axis 4, then:
/1A1000R	Moves Axis 4 to absolute position 1000
/1?0	Retrieves Axis 4 motor position
/1L10R	Sets Axis 4 acceleration to 10.

Motor sequencing in multi-axis commands

By default, motors responding to multiple multi-axis commands in a long string will wait until all motors have completed their motion before moving on to the next command in the string.

In linear interpolation mode the motors are coordinated to reach the same point at the same time. Eg for drawing straight lines. Issue /1*an65536*R to turn on linear interpolation. Implementation date TBD. */1an65536,65536,0,0R sets* Axes 1 and 2 into linear interpolation mode for drawing straight lines. */1an0,0,65536,65536R sets* Axes 3 and4 into linear interpolation mode for drawing straight lines. */1an65536,65536,65536,65536R sets* Axes 1 and 2 into linear interpolation mode and separately sets axes 3 and 4 into interpolation mode Issue /1an0 to turn off linear interpolation mode.

Making on-the-fly parameter changes (*Immediate* Commands)

Overview

Some parameters can be changed "on the fly" (i.e. while another command is executing) using Immediate commands. This can be done for one axis (currently selected axis) or for all four axes on the EZ4AXIS.

NOTE: The run command (*R*) is not required for on-the-fly commands, but does not affect operation if added.

Examples

- */1V2000* (no *R* required if in motion) will change the velocity of the currently selected axis.
- */1V1000,2000,3000,4000* will change the velocities of all four axes while in motion.
- */1A1000,2000,3000,4000* will change the targets of all four axes while in motion.
- */1A1000,,1000* will change the target position for axes 1 and 3 while in motion.

On-the-fly changes allow truly independent control of the motors, as opposed to a stored program mode where axes will wait for each other to reach a coordinate.

Note that these commands must be issued one at a time. So, for example, */1V1000V3000* will ignore the *V3000* if the unit is running.

Immediate Command list

A	Absolute position
P	Positive relative move
D	Negative relative move
V	Velocity
L	Acceleration factor
m	Move current
n	Select various modes
J	Turn power output drivers on/off (typically used for solenoids)

The Immediate commands are also listed and described in Table 1, Command Set, which begins on page 71.

Analog and Digital Inputs

The EZCTRL17XR has sixteen analog inputs which can be read back as analog values. Eight of these are parameters of the channel such as current, and the remaining eight of these are routed to the I/O connector as 2 per connector.

Read back digital IO input values on the 10 Pin I/O Connector (/1*?41 /1?42 /1?43 /1?44* command)

?41 reads back the digital values associated with 10 pin I/O connector on channel 1 in order UpperLimit, Home,ADC2,ADC1 where the ADC is considered high or low depending on the threshold set by the "at" command (see "at" command below).

The result is a number ranging from 0-15, representing a 4-bit binary pattern in which:

Bit 0 = ADC1 (Switch 1) Bit 1 = ADC2 (Switch 2)
Bit 2 = Home (Opto 1) Bit 3 = UpperLimit (Opto 2)

Example: Readback number is 9, which is equivalent to digital 1001. This indicates Opto 2 high. Opto 1 low; Switch 2 low; Switch 1 high.

Read back analog IO input values (the *?aa1* command)

The *?aa1* command reads back the two analog ADC values on the **10 Pin** IO connector and the two ADC's measuring motor currents on the drive. inputs in this order 4, 3, 2, 1, which are input 4 (From Drive), input 3 (From Drive), ADC 2, and ADC1 respectively.

Example: */1?aa* 1

Response: *8767,6544,6943,10720 No Error*. These are the values of inputs 4, 3, 2, and 1 respectively, expressed by a number from 0-65535 that represents a linear scale of 0-3.3V.

Additional */1?aa1* or */1?aa2* or */1?aa3* or */1?aa4* commands read back the analog values on the limit/home inputs for the respective axis

Programmable Threshold on Analog Readback (the *at* command)

at Adjust thresholds at which a high or a low is called, for the /?41 comand, on the two analog inputs on the 10 Pin I/O connector..

Thresholds are expressed as five-digit numbers in a range from 00000-65535, which linearly represents the 0-3.3V input. The default threshold value is 24200 (1.22V).

Example: */1at3209999R*. This sets the threshold on Axis 3 ADC2 to 09999 (0.5 volts). Key components:

at Threshold command

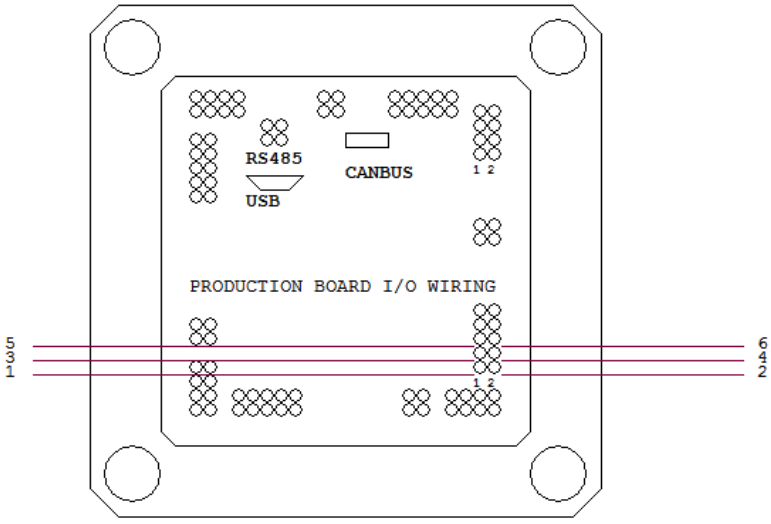
32 Axis/input identifier, in this case Axis 3 and Limit Input 2 (upper limit). The full range is 11,12,21,22,31, 32,41,42.

09999 Threshold value, 00000–65536, representing 0–3.3V. (Threshold value = (desired threshold voltage/3.3) x 65535.) Default is 24200 (1.22V).

The threshold value for limits must always be **entered as five digits**. Thus it is necessary, in this example, to insert a leading zero after the axis/input identifier to reach the required number of digits.

?aat Read thresholds of all ADCs on the four 10 pin I/O connectors on the drive. The order of readback (in terms of the axis/input identifiers explained above) is

LEFT END Axis1ADC2, Axis1ADC1, Axis2ADC2, Axis2ADC1, Axi3ADC2, Axis3ADC1, Axis4ADC2, Axis4ADC1. *RIGHT END*
Example readback /1?aat :
6144,6144,6144,6144,9999,6144,6144,6144 No Error



5 GND 6 GND
3 OPTO 1/HOME/LOWER LIMIT/DIGITAL IN 2^2 4 OPTO 2/UPPER LIMIT/DIGITAL IN 2^3
1 SWITCH 1 IN/DIGITAL 2^0/ ANALOG CH1 2 SWITCH 2 IN/DIGITAL 2^1/ ANALOG CH2

Figure 2 EZ4AXIS17XR Input Connections

Input-Dependent Basic Operations

Halt and wait for IO or Limits (the *H* command)

Motors may be halted, to wait in response to the status of either the Digital/Analog IO switches or the Limit switches before executing a command.

For this purpose, the *H* command is followed by a three-digit operand, The operands specify which input and what polarity the axis will wait for after halting.

- Three-digit operands apply to the inputs on the 10-pin I/O connectors. The most significant digit is axis, second digit is polarity (high or low), and the third digit is limit 1 or 2:

  101 wait for low on Axis 1 ADC1
  111 wait for high on Axis 1 ADC1
  102 wait for low on Axis 1 ADC2
  112 wait for high on Axis 1 ADC2
  103 wait for low on Axis 1 limit 1 (lower)
  113 wait for high on Axis 1 limit 1 (lower)
  104 wait for low on Axis 1 limit 2 (upper)
  114 wait for high on Axis 1 limit 2 (upper)

  201 wait for low on Axis 1 ADC1
  211 wait for high on Axis 1 ADC1
  202 wait for low on Axis 1 ADC2
  212 wait for high on Axis 1 ADC2
  203 wait for low on Axis 1 limit 1 (lower)
  213 wait for high on Axis 1 limit 1 (lower)
  204 wait for low on Axis 1 limit 2 (upper)
  214 wait for high on Axis 1 limit 2 (upper)

  301 wait for low on Axis 1 ADC1
  311 wait for high on Axis 1 ADC1
  302 wait for low on Axis 1 ADC2
  312 wait for high on Axis 1 ADC2
  303 wait for low on Axis 1 limit 1 (lower)
  313 wait for high on Axis 1 limit 1 (lower)
  304 wait for low on Axis 1 limit 2 (upper)
  314 wait for high on Axis 1 limit 2 (upper)

401 wait for low on Axis 1 ADC1

411 wait for high on Axis 1 ADC1

402 wait for low on Axis 1 ADC2

412 wait for high on Axis 1 ADC2

403 wait for low on Axis 1 limit 1 (lower)

413 wait for high on Axis 1 limit 1 (lower)

404 wait for low on Axis 1 limit 2 (upper)

414 wait for high on Axis 1 limit 2 (upper)

NOTE: If an edge detect is desired, a look for low and a look for high can be placed adjacent to each other, e.g., *H301H311* is a rising-edge detect.

Examples:

/1H101P100R Halt and wait for low on Axis 1 Limit input 1 (lower limit) and then move positive 100 microsteps.

/1gH211P100GR (Looping example) Halt and wait for high on Axis 2 Limit input 1 (lower limit) and then move positive 100 microsteps. This repeats infinitely due to the loop created by *g* and G

NOTE: Issuing */1R* will also break through a halt.

NOTE: Issue /1T to Terminate

Skip and Branch on IO or limits (the *S* and *e* commands)

You can program the EZ to skip an instruction within a command string and branch to another command string stored in EEPROM.

Overview

- Skipping is implemented using the *S* command, which is accompanied by three-digit operand (as described for the *H* command, above), indicating the condition at one of the inputs.

  For example, *S102* means "Skip next instruction if there is a low on Axis 1 limit 2 (upper limit)."

  Or, *S112* means "Skip next instruction if there is a high on Axis 1 limit 2 (upper limit)."
- Branching is implemented with the *e* or execute command, which tells the drive to run the command string in a specific memory location 0-15. For example, *e2* means run the string residing in location 2. Note that e2 is a GOTO, not a GOSUB.

Example

The following example stores two command strings in EEPROM locations 0 and 1, and the program skips an instruction and switches from one string to the other depending on the state of input 3.

Send these command strings:

/1s0gA0A10000S112e1G0R (stored string 0, executed at startup)

/1s1gA0A1000S102e0G0R (stored string 1)

Stored string 0 command breakdown:

/1	Talk to device number 1 on the EZ bus (Drive 1).
s0	Store following in memory location 0 (executes on power-up).
g	Start loop.
A0	Go to absolute position 0.
A10000	Go to absolute position 10000.
S112	Skip next instruction if 1 (high) on axis 1 input 2
e1	Jump to string 1 (execute command string stored in memory location 1.) (This is the branch operation.)
G0	End of loop (*0* indicates infinite loop).
R	Run.

Stored string 1 command breakdown:

/1	Talk to device number 1 on the EZ bus (Drive 1).
s1	Store what follows in memory location 1.
g	Start loop.
A0	Go to absolute position 0.
A1000	Go to absolute position 1000.
S102	Skip next instruction if 0 (low) on axis 1 input 2.
e0	Jump to string 0 (execute command string stored in memory location 0.) (This is the branch operation.)
G0	End of loop (0 indicates infinite loop).
R	Run.

Result: At power-up, the code will cycle the motor between position *A0* and *A10000* if input 2 is high, and between *A0* and *A1000* if input 2 is low.

NOTE: Loops can be exited by storing nothing in a memory location and skipping to that location.

For example, the command */1s14R* stores nothing in memory location 14, and a skip to that location (eg S112e14) will exit the loop.

Advanced looping techniques

Create a nested loop

The following example shows how to construct a nested loop, or loop within a loop. Nested loops can be up to four levels deep.

Send this command string:

/1aM2gA1000A10000gA1000A10000G10G100R

Command breakdown:

/1	Talk to device at Address 1 on the EZ bus.
aM2	Talk to Axis 2.
g	Start outer loop.
A1000	Go to absolute position 1000.
A10000	Go to absolute position 10000.
g	Start inner loop.
A1000	Go to absolute position 1000.
A10000	Go to absolute position 10000.
G10	Repeat inner loop 10 times. (end of Inner loop).
G100	Repeat outer loop 100 times. (end of outer loop).
R	Run.

Result: The first *g* (lower case) command starts the outer loop, and the second *g* command starts the inner loop. The first *G* (upper case) command terminates the inner loop and specifies how many times it is run, while the second *G* command does the same for the outer loop.

To create an endless nested loop, change the second *G* command from *G100* to *G0* (zero). The nested loop will now run until interrupted by the *T* command, e.g. */1T*. (Do not use */1T2*, */1T3* etc. to terminate a loop since the behavior is undefined and may change in the future.)

Set up standalone operation

Standalone operation is implemented by storing the appropriate command string in EEPROM memory location 0. The command string in this location runs automatically at power-up. Examples of this are shown on pages 39 and 40.

Readjusting velocity, acceleration, and current

If the motor behavior is problematic using the standard values for the EZ4AXISXR17 provided in Section 3, here are some tips for making corrective adjustments to velocity, acceleration, and current.

Command	Description
V	Velocity: reduce velocity if the motor stalls, or try adjusting *L* or *m* as described below. Maximum achievable velocity depends on the available supply voltage. Also, move current may need to be adjusted to obtain a desired velocity. Standard setting is *V1000*.
L	Acceleration (acceleration factor): depends on adequate supply power and voltage. If available power is insufficient, back off acceleration. Increase voltage, if possible, to overcome the motor's back emf. Standard setting is *L10*. The EZAXIS17XR equation for acceleration is: Acceleration (in microsteps / sec^2) = (L value) x TBD For example, if V=10000 and L=1, it will require TBD seconds to reach final velocity. The EZ4AXIS17XR equation for acceleration is TBD.
m	Maximum move current: more current provides more force and more acceleration. If motor stalls, try increasing this value. Given as a percentage of the maximum output driver current, which is 2A for the EZ4AXIS17XR and 0.5Afor the EZ4AXIS17XR. Standard setting is *m30* (30% of 0.5A).
h	Hold current: increase if motor is slipping when attempting to hold a position. Given as a percentage of the maximum output driver current, which is 0.5Afor the EZ4AXIS17XR. Standard setting is *h10* (10% of 0.5A).

For example, if a motor doesn't move and makes a hammering sound, it is stalled. Decreasing *V* or *L* may be needed. Move current (m) may also need to be increased or supply voltage may need to be increased.

Note that if a another cause of a motor buzzing but not moving is that only one phase is connected or one of the phase drivers is blown. Another symptom of only one phase being connected is that the motor may spin in a random direction with low torque.

Single-Axis programming supplemental examples

The following single-axis programming examples are provided to supplement the preceding instructions.

Example #1 (loop: moves with waits)

/1aM2gA10000M500A0M500G10R

Command breakdown:

/	Start character. Tells the EZSteppers® that a command is coming in.
1	Drive 1 (device address, set via address switch on device).
aM2	Axis 2
g	Start a repeat loop.
A10000	Turn to absolute position 10000.
M500	Wait for 500 milliseconds.
A0	Turn to absolute position 0.
M500	Wait for 500 milliseconds.
G10	Repeat string 10 times beginning from the location of the *g* command.
R	Run the command.

NOTE: To terminate the above loop while in progress, issue the *T* command, e.g., */1T2* for Axis 2. This terminates any programs running on the drive card. (Do not use */1T2*, */1T3* etc. to terminate a loop since the behavior is undefined and may change in the future.)

Example #2 (loop: set current, wait for Switch 2 closure, go home)

NOTE: This is a standalone operation example.

/1s0aM1m75h10gJ15,15,15,15M500J0,0,0,0M500G10H104A1000A0Z10000R

Command breakdown:

/1s0	Stores the program that follows in EEPROM location 0 (string 0 is executed on power-up).
aM1	Select Axis 1.
m75	Set move current to 75% of max.
h10	Set hold current to 10% of max.
g	Start a loop.
J15,0,0,0	Turn on all on/off drivers for Axis 1.
M500	Wait 500 ms.
J0,0,0,0	Turn off both on/off drivers for Axis 1

M500 Wait 500 ms.

G10 Repeat loop above 10 times.

H104 Halt and wait axis 1 input4 to go low.

A1000 Move to absolute position 1000.

A0 Move to position 0.

Z10000 Home the stepper. Maximum steps allowed to find opto is set to 10000.

R Run.

Example #3 (loop: monitor four switches and execute four different programs depending on which switch input is pushed)

NOTE: This is a standalone operation example, which stores five command strings for an endless loop. This loop runs automatically at startup, because it begins at the program *0* location.

Setup

/1s0aM3gS111e1S112e2S113e3S114e4G0R
Stores command string in EEPROM location 0 (string 0 is executed on power-up). (S111= skip next instruction if high on axis input 1.)

/1s1A1000e0R Stores command string in EEPROM location 1.

/1s2A2000e0R Stores command string in EEPROM location 2.

/1s3A3000e0R Stores command string in EEPROM location 3.

/1s4A4000e0R Stores command string in EEPROM location 4.

Execution

- At power-up, string *0* automatically executes on Axis 3 and loops around sampling each switch one by one (S111, etc.), and skipping the subsequent instruction if it is not depressed.
- If a switch—for example Axis 1 Input 1—is depressed, string 1 is executed, which moves the stepper to absolute position 1000.
- Execution then returns to string 0, due to the *e0* command at the end of the string.
- If the switch is still depressed it will jump to string 1 again, but since the motor is already at that position there will be no visible motion.
- If another switch is closed, the program will also jump to that stored string.

To terminate this endless loop, issue */1T*. This stops all commands executing on device with address 1.

NOTE: Using an *e* command to go to another program is more of a "GOTO" than a "GOSUB" since execution does not return to the original departure point .

Example #4 (loop: move 1000 steps forward on rising edge of Axis 1 Input 1)

/1aM2gH101H111P1000G0R

The endless loop halts and first waits for a 0 level on Axis 1 Input 1, then waits for a high on Axis 1 Input 1.

Then a relative move of 1000 steps is issued, and the program returns to the beginning to look for another rising edge.

NOTE: To terminate the above loop while in progress, issue */1T*.

Example #5 (loop: use threshold setting to regulate pressure)

It is possible to regulate pressure by turning a pump on or off depending on an analog value read back, by designating the threshold of the one/zero call as the regulation point.

/1at3208000gS302P1000G0R.

This command first sets a threshold level using the *at* command. Then it starts an endless loop during which it responds to a high on axis 3 input 2 by moving the motor 1000 microsteps positive. As long as axis 3 input 2 remains low, it skips the move command (*P*).

Multiple-axis/multiple drive supplemental examples

The following examples utilize multi-axis commands.

Example #5: coordinated motion with all cards on a bus, and all motors on each card, performing same motion

This example also shows how to address all drives and all axes on a bus.

/_A1000,1000,1000,1000R

Command breakdown:

/_ (Slash then underscore) Select all drives on bus.

A1000 Go to absolute position 1000. The four comma-delineated positions above represent the same command applied to the four axes on the drives.

R Run. All motors on all drives go to absolute position 1000.

Example #6: coordinated motion between two separate drive cards.

This example also shows how to set up commands without running and then later send a Run command separately so that multiple motors/drives start and run at the same time (within 10mS).

NOTE: The following are multi-axis commands.

/1A1000,200,300,400	Set up drive card 1 Axis 1 to absolute position 1000; Axis 2 to position 200; Axis 3 to position 300; and Axis 4 to position 400.
/2A200,300,400,1000	Set up Drive card 2 Axis 1 to absolute position 200; Axis 2 to position 300; Axis 3 to position 400; and Axis 4 to position 1000.
/AR	Run current commands in buffer for all axes on Drive card 1 and Drive card 2. The /A command addresses a bank defined as Drives 1 and 2. (Banks are described in Appendix 1, Addressing Methods Reference.)

Example #7: synchronized motion among different drives

This example also shows how to set up commands without running and then later send a Run command separately so that multiple motors/drives start and run at the same time (within 10mS).

/1aM3A1000	Set up Drive 1 Axis 3 to move to absolute position 1000.
/2aM1A100	Set up Drive 2 Axis 1 to move to absolute position 100.
/AR	Run commands on Drive 1 Drive 2. The */A* command addresses a bank defined as Drives 1 and 2. (Banks are described in Appendix 1, Addressing Methods Reference).

Example #8: select drives in bank, then issue command to bank

If no axis is indicated in any command, the command is issued to the last axis specified on each drive in the addressed bank. So axes can be pre-selected prior to issuing the bank command, for example:

/1aM3R	Select drive card 1 Axis 3.
/2aM1R	Select drive card 2 Axis 1.
/AP1000R	Move drive card 1 Axis 3 and drive card 2 Axis 1 relative 1000 microsteps positive. The /A command addresses a bank defined as Drive cards 1 and 2. (Banks are described in Appendix 1, Addressing Methods Reference.)

5. Limits (*n2*)

Overview

Each axis can be set up to stop motor rotation when a mechanical limit is signaled by a switch closure or opening. Limits are not active by default, and must be specifically turned on. */1n2R* or */1n2,2,2,2R*

When a limit is engaged, the motor will not respond to a move command in the direction in which the limit is engaged, but will perform a move in the direction that backs out of the limit.

NOTE: It is necessary to turn off limits while homing, because the limits will inhibit a correct homing position from being achieved.

Hookup

Make connections at the Limit/home connector for the relevant axis (see figure below). Each 6-pin limit/home connector provides two inputs: an upper limit and a lower limit. The upper limit operates in the positive movement direction, while the lower limit operates in the negative movement direction. The lower limit is also the homing input, as determined by programming.

Status of limits are read back by the */1?41 , /1?42, /1?43* and */1?44* command.

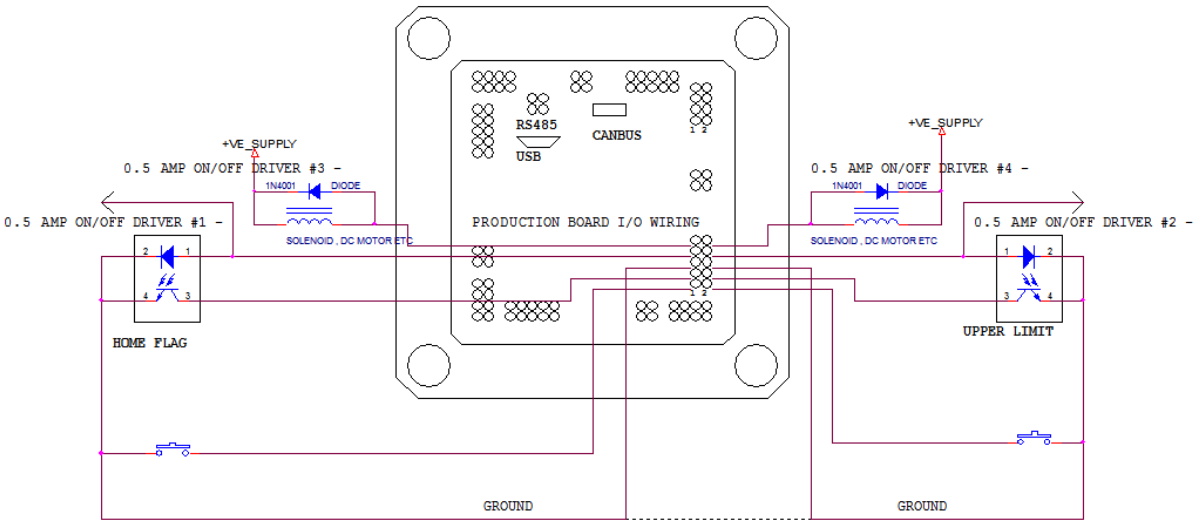


Figure 3 Limit Connections

The inputs may use either optical or mechanical switching devices, and include outputs for power LEDs. These outputs are internally connected to +5V through 200-ohm resistors. For sensors such as Hall sensors that need direct access to +5V the 200-ohm resistors may be shorted across.

500mA total is available across all 5V connections (Encoder + LEDS).

NOTE: The inputs are relatively high impedance at 10K ohms and will pick up noise if bundled with the motor wires, etc. For long cable runs, each input line should be shielded. The addition of a 0.1µF ceramic capacitor from the input to ground at the board connector may be an alternative to shielding, but could slow the response.

Basic limits setup

As needed, refer to "Commands for limits," below, for additional clarification.

1. Set up mechanical assembly with limit switch or switches placed in desired location(s).
2. Activate limits on desired axis by issuing the *n2* command, e.g., */1aM2n2R*. Or use multi axis /1n2,2,0,2R
3. Ensure that a positive move, e.g. */1aM1P1000R,* moves toward the upper limit and away from the lower limit. If the motor moves in the wrong direction, reverse the connections to only one of the windings of the motor. Inputs can be read by /1?41 etc
4. Set limit polarity if necessary:

   The default condition expects the limit switch to be low when away from the limit (as is the case when an optical switch is used). If the limit switch is to be high when away from the limit (as with a normally-open switch), issue the command *f1* to reverse the polarity that is expected. This can be done per axis; for example, */1aM1f1aM2f0R* selects different polarities for the limits of Axes 1 and 2. So *f1* = normally open, and *f0* = normally closed. Or use multi axis */1f1,0,1,1R* etc.
5. Set limit input threshold if needed with the *at* command (for example, because some sensors will not pull to a TTL low level when closed).

   To calculate the threshold number to be used in the command: (Desired threshold voltage/3.3) x 65535 = threshold number. Example for 2.00 volts: */1at2243690R,* where *22* indicates Axis 2 input 2 (upper limit) and *43690* is the threshold number for 2.00 volts according to the formula above (note this number must always be 5 digit, use leading zeros as necessary.

   Confirm thresholds with the *?aat2* command.
6. Issue move commands and confirm that the motor stops when it reaches the limit or limits that have been set up.

Commands for limits

n2 Makes limits active on a per-axis basis, e.g., */1aM3n2R* activates limits on Axis 3.

f Set limits polarity. Set expected limit switch to be normally open or normally closed. Normally open is *f1* and normally closed is *f0*. The default is normally closed. Normally closed is generally preferable because opening a switch can interrupt current to terminate motion immediately.

at Adjust thresholds, if needed. For example, the sensor used for limit switching may not pull to a full TTL level, as is the case with a reflective sensor. The default threshold is 24200 (1.22V).

Thresholds are expressed as five-digit numbers in a range from 00000-65535, which linearly represents the 0-3.3V input. The default threshold value is 24200 (1.22V).

Example: */1at3243690R*. This sets the threshold on Axis 3 input 2 (upper limit) to 43690 (2 Volts). Key components:

at Threshold command

32 Axis/input identifier, in this case Axis 3 and Limit Input 2 (upper limit). The full range is 11,12,21,22,31, 32,41,42.

43690 Threshold value, 43690, representing 0–3.3V. (Threshold Voltage = 3.3V x(43690/65536.)

The threshold value for limits must always be entered as five digits. Thus it is necessary, in this example, to insert a leading zero after the axis/input identifier to reach the required number of digits.

To set multiple thresholds, repeat the *at* command for each threshold to be set:

1at1115952at2105952at1211904at2211904R.

?aat Read thresholds of all home/limit inputs on the drive. 12, 11, 22, 21, 32, 31, 42, 41, where 21, for example, indicates Axis 2 input 1Example readback:
6144,6144,6144,6144,9999,6144,6144,6144 No Error

NOTE: Input 1 is Lower Limit/Home, and input 2 is Upper Limit. The response is a number from 00000-65536, which represents the 0-3.3V range available at each input.

6. Emergency stop / kill move

Overview

This command still to be implemented.

7. Digital IO (switch-controlled) applications

The EZ4AXIS can respond to on/off states applied to the Analog/Digital IO 8-pin connector. A switched input can be used to notify an axis when a specific mechanical position is reached, or for any other practical purpose requiring an on/off signal, such as:

- Halt; then move or execute another program stored in memory
- Skip the next step
- Wait for another change in status of the input
- Monitor an input using an endless loop, or base an action on the status of an input while running the loop a specified number of times.

Hookups

The 10-pin Analog/Digital IO connector provides a set of four multipurpose inputs, accessible by any of the four axes via programming. All inputs operate on 0 to 3.3V level signals, although thresholds can be adjusted to accommodate non-TTL level switch closures.

Status of digital inputs are read back by the */1?41 , /1?42, /1?43* and */1?44* command.

The four digital inputs consist of 2 analog inputs thresholded with the *at* command to give two digital inputs and two LV TTL level digital inputs.

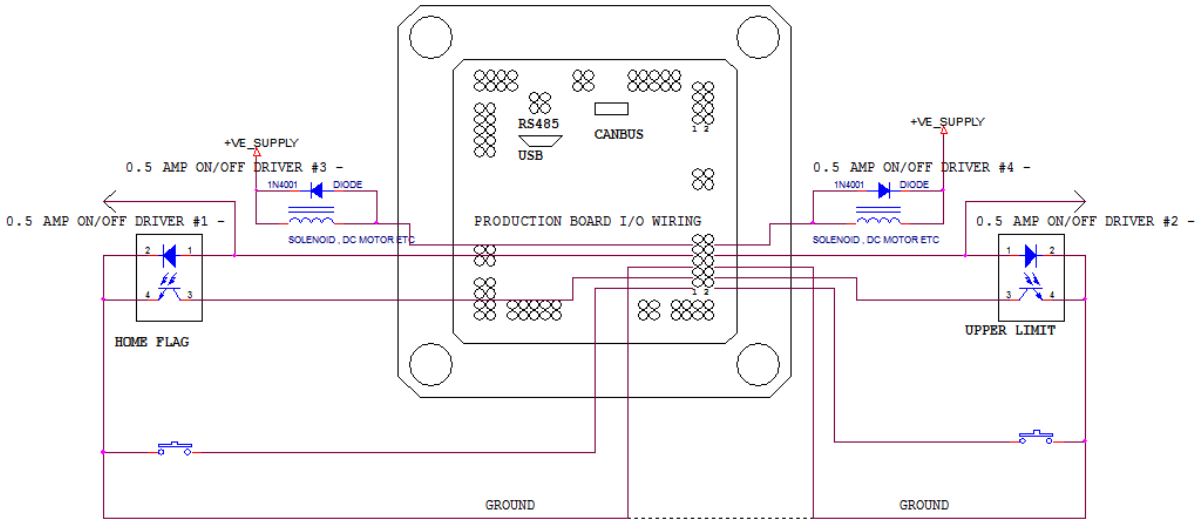


Figure 4 Switch inputs for the EZ4AXIS17XR

The inputs may use either optical or mechanical switching devices, and include outputs for power for LEDs. These outputs are internally connected to +5V through 200-ohm resistors. For sensors such as Hall sensors that need direct access to +5V, the 200-ohm resistors may be shorted across. (Contact factory for removal instructions)

500mA total is available across all 5V connections (Encoder + LEDS).

Optical or mechanical switches may be used on any input. If four optical switches are desired for the IO connector, power for the additional optical switches can be drawn from the 5V supply pin on one of the encoder connectors. This power may require an external resistor in series with the LED in the optical switch.

NOTE: The inputs are relatively high impedance at 10K ohms and will pick up noise if bundled with the motor wires, etc. For long cable runs, each input line should be shielded. The addition of a 0.1µF ceramic capacitor from the input to ground at the board connector may be an alternative to shielding, but could slow the response.

Commands

?aa Read back ADC values (values after analog/digital conversion) on specified input *(/1?aa1*= Axis 1*, /1?aa2*=Axis 2, etc.*)*. These values are on a scale of 0 -65535 as the input varies from 0–3.3V. The inputs as shipped have a resolution of about 7 bits, but can be improved to exceed 10 bits with the removal of the input overvoltage protection circuitry (call AllMotion for details).

at Adjust thresholds. Thresholds can be set for on/off detection, since IO connector inputs are essentially analog. The default threshold value is 24200 (1.22V).

Example: */1at309999R*. This sets the threshold on input 3 to 09999 (2.02V).

3 Input identifier, in this case Input 3 (Opto 1).

09999 Threshold value, 00000–65536, representing 0–3.3V. (Threshold value = (desired threshold voltage/3.3) x 65536.)

Note that it is necessary to insert a leading zero after the input identifier (3), since the threshold value must always be entered as five digits (00000-65536).

NOTE: In prior firmware versions, all thresholds were simultaneously programmed when input 4 on the 8-pin IO connector was set using the */at4XXXXXR* command.

?aat Read back thresholds for all eight inputs on the 10-pin connectors. The readback order is inputs 4, 3, 2, 1.

H Halt

S Skip next command

M Wait

e Execute command string stored in specified EEPROM memory location. Used for branching in conjunction with halts, skips, and waits. E.g., */1e2* means "execute contents of memory location 2."

Examples

For programming examples, see TBD "**Error! Reference source not found.**" beginning on page TBD **Error! Bookmark not defined.** and "Single-Axis programming supplemental examples" beginning on page 39.

8. Analog control applications

This section describes analog control applications that use one or more of the inputs on the 8-pin Analog/Digital IO Connector.

These applications utilize potentiometers as the input devices. However, any input ranging from 0–3.3V will be accepted.

Analog inputs

The four inputs on the 8-pin connector are all ADC.

The ADC values, representing the potentiometer positions, can be read using the *?aa* command *(/1?aa1 =* axis 1*, /1?aa2 =* axis 2, etc.*)*. These values are on a scale of 0 - 16368 as the input varies from 0 - 3.3V. The inputs as shipped are good to about 7 bits, but can be made to be better than 10 bits with the removal of the input overvoltage protection circuitry (contact AllMotion for instructions). The EZ4AXIS ignores all voltages higher than 3.3V.

Noise considerations

The inputs are relatively high impedance at 10K ohms and will pick up noise if bundled with the motor wires, etc. For long cable runs, each input line should be shielded. The addition of a 0.1µF ceramic capacitor from the input to ground at the board connector may be an alternative to shielding, but could slow the response.

9. Homing

Overview

Homing provides a known starting point for each motor, from which each operation can begin, and from which absolute positions are calculated. There are two homing modes:

- N1 Mode, homing to a TTL input on the 6-pin Home/Limit connector for that axis. This is the default homing mode. Typically a mechanical or optical switch is set up to open or close when the motor reaches the home position. See "Homing to opto/switch (default *N1* mode)," below.
- N2 Mode, homing to an index. In this mode, the index pulse from an encoder is used for homing. For index homing, see "Homing to encoder index (N2 mode)," below.

Homing to opto/switch (default *N1* mode)

Hookups

Each axis on the drive has its own set of limit/home inputs on 10-pin connectors, as shown below. The lower limit input is also the home input. Status of home/limit inputs are read back by the */1?41 , /1?42, /1?43* and */1?44* command.

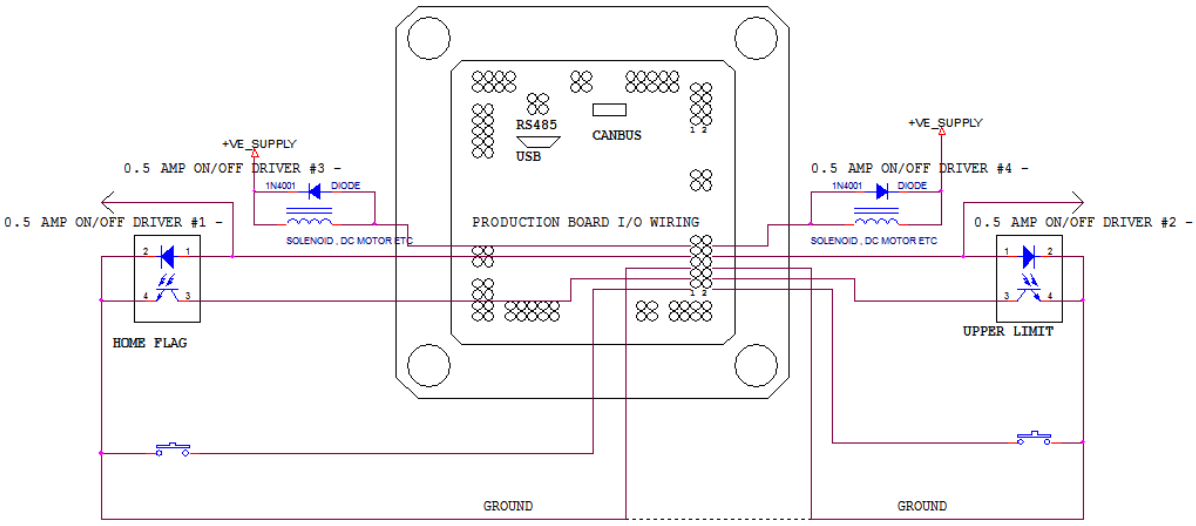


Figure 5 Home/Limit Connections production drive (different on prototype model)

These connections can be used with any device that sends out TTL level signals. Typically optical or mechanical switching devices, and include power outputs for optical LEDs. The High Low threshold can also be changed using the *at* command. Further since each input has a pullup resistor built into the board, a simple switch to ground can be used as a home switch. See the wiring diagram on the AllMotion website EZ4AXIS product page, for more detail if needed.

Noise considerations

The inputs are relatively high impedance at 10K ohms and will pick up noise if bundled with the motor wires, etc. For long cable runs, each input line should be shielded. The addition of a 0.1µF ceramic capacitor from the input to ground at the board connector may be an alternative to shielding, but could slow the response.

Homing behavior

The *Z* command initializes the motor to a known position, called Home (the Home position is usually aligned with a switch closure or opening). This is position 0. All absolute positions are relative to the Home position. When the *Z* command is issued, the motor turns toward position 0 until the home switch is interrupted. If the switch is already interrupted, the motor will back out and return until the switch is re-interrupted.

Basic homing setup

Refer to "Commands for homing to opto/switch," below, as needed for additional clarification.

NOTE: Homing occurs at the speed currently set by the *V* (velocity) command. For accuracy and reliability, it is important to home at a slow speed. If high-speed homing is desired, home at high speed first and then home again at low speed.

1. Make sure switch and flag are set up to be unambiguous. For example, when the motor is at one end of travel, the home flag should interrupt the switch; and when at other end of travel, the home flag should not interrupt the switch. There should be only one zero-to-one transition possible on the home input in the whole range of motor rotation. Do not use a tiny flag that you can get on the wrong side of.
2. Ensure that a positive move, e.g. */1aM1P1000R,* moves away from home and the home flag. If the motor does not move away from home on a positive move, reverse the connections to only one of the windings of the stepper motor. (Do not use *f* command to fix this)
3. Set home flag polarity if necessary. Again, before doing this step make sure the *P1000* command moves away from home.

   The default condition expects the home flag to be low when away from home (as is the case when an optical switch is used). If home flag is to be high when away from home (as in the case of a normally-open switch), issue the command *f1* to reverse the polarity

that is expected of the home flag. This can be done per axis; for example, */1aM1f1aM2f0R* selects different polarities for the home flags of Axes 1 and 2. So *f1* = normally open, and *f0* = normally closed. Or use multi axis command */1f1,0,1,1R* etc.

NOTE: If the home flag polarity is set incorrectly, or if the connections to the motor are reversedthe motor may move in the wrong direction when attempting to home.

4. Set home input threshold if needed with the *at* command; confirm with the *?aat* command: (Desired threshold voltage/3.3) x 65536 = threshold number. Example for 2.00 volts: */1at3109920R,* where *31* indicates Axis 3 input 1 and *09920* is the threshold number according to the formula above (a zero is added at the beginning because this must be entered as five digits).
5. Issue the *Z* command with the desired maximum number of steps, for example */1aM1Z100000R*. Or with the home flag polarity setting included, for example */1aM1f1Z100000*. Observe motor behavior.

Commands for homing to opto/switch

N1 Enter the Homing to Switch mode. This is the default mode, so it is not usually necessary to issue this command.

Z The homing command. The maximum number of steps allowed to go toward home is defined by the *Z* command operand, e.g., *Z4000*.

f Set Home polarity. Set expected home switch to be normally open or normally closed. Normally open is *f1* and normally closed is *f0*. The default is normally closed. Normally closed is generally preferable because it is "fail safe" in case of disconnection of the sensor.

NOTE: If the *f* command is not set correctly, the motor may move in the wrong direction when homing.

at Adjust thresholds, if needed. The default is 1.24V.

For example, consider a home sensor that doesn't fully pull to a TTL low level (e.g., a reflective sensor). The threshold can be adjusted to accommodate the sensor's output level without external signal conditioning.

Thresholds are expressed as five-digit numbers in a range from 00000-65535, which linearly represents the 0-3.3V input. The default threshold value is 24200 (1.24V).

See example next page.

Example: */1at31xxxxxR*. This sets the threshold on Axis 3 input 1 (lower limit/home) to xxxxx (.

Key components:

31 Axis/input identifier, in this case Axis 3 and Input 1. The Home input is always Input 1.

xxxxx Threshold value, 00000–65536, representing 0–3.3V. (Threshold value = (desired threshold voltage/3.3) x 65536.)

The threshold value for home/limits must always be entered as five digits. Thus it is necessary, in this example, to insert a leading zero after the axis/input identifier to reach the required number of digits.

?aat (Requires firmware version 7.50 or higher.) Read thresholds of all home/limit inputs on the drive. The order of readback (in terms of the axis/input identifiers explained above) is 12, 11, 22, 21, 32, 31, 42, 41.
Example readback:
6144,6144,6144,6144,9999,6144,6144,6144 No Error

NOTE: Input 1 is Lower Limit/Home, and input 2 is Upper Limit.

Homing to encoder index (*N2* mode)

NOTE: Homing should be done at a slow speed, especially if homing to a narrow index pulse on an encoder, which could be missed at high speeds.

Overview

In this mode, the specified axis (Axis 1 or 2) will home to the index of the encoder associated with the axis. Note that the index is sampled at a 20kHz rate, and may be missed if motor velocity is too high. Velocity must be such that the index is at least 100µs in width.

Hookup

For encoder hookup, refer to Section 10, "Using encoders," page XX TBD

Commands

N2 Enter the Home to Encoder Index mode.

Example

/1aM1N2Z10000R	Home Motor1 to Encoder1 Index
./1aM2N2Z10000R	Home Motor2 to Encoder2 Index.

10. Using encoders

Overview

Use 8 pin encoder input provided. Encoders may be used in the following ways:

- **Encoder Following:** Motor follows the count generated by one of the two auxiliary encoder inputs.
- **Position Correction:** Motor position is automatically corrected if wrong position is detected. This is done by the encoders dedicated to each of the motor.
- **Auto recovery in Position Correction mode:** auto recovery scripts are run if Position Correction cannot correct a motor position.
- **Overload report mode:** This mode verifies the current position against the encoder and reports any deviation, but does not attempt to auto correct the error.
- **Homing:** provide a starting point for the motor using encoder index pulses. This mode is explained in "Homing to encoder index (N2 mode)" on page 55.

Encoder hookup

There are four encoder input connectors, The connectors accept index and AB quadrature pulses, and provide 5V power to the encoders. See Figure 6 below. (Also has optional absolute encoder capability which is not implemented yet)

Electrical Notes:

- 500mA total is available across all 5V connections (Encoder + LEDS).
- Encoders must have 0.2V low to >4V high at the connector.
- Can operate with 3.3V encoder with factory modification.
- Max Encoder frequency is 4MHz
- Contact factory for optional absolute encoder connection.
- Refer to wiring diagram also.

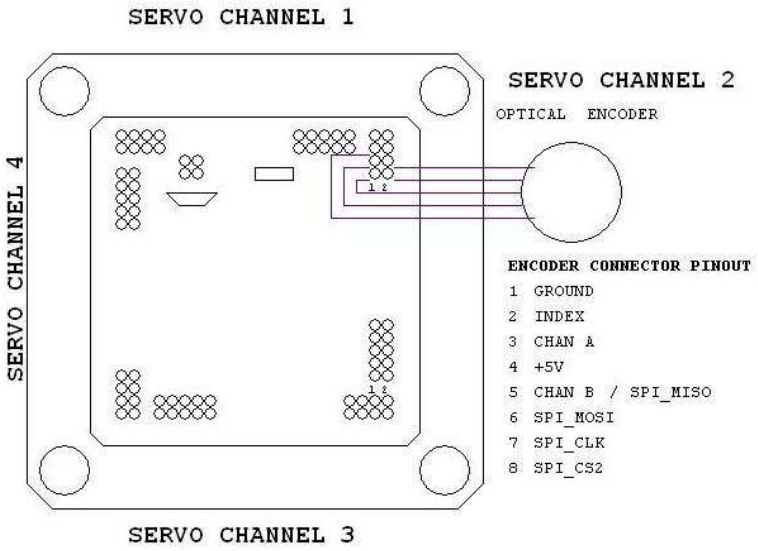


Figure 6 Feedback Encoder Connectors

NOTE: Cable from encoders should be shielded, especially in environments with significant noise. They will pick up noise if bundled with motor wires, etc. For long cable runs, each input line should be individually shielded.

Encoder following mode (*n64*)

NOTE: THIS FUNCTION IS NOT IMPLEMENTED YET IN THE EZ4AXISXR YET –USE EZ4AXIS FOR THIS FUNCTION

NOTE:

This mode can be used with either Axis 1 or Axis 2, which have encoder inputs.

Overview

In this mode, also called "Slave to Counter," the motor takes the AB count from an encoder not coupled to the motor, and uses this count as a commanded position. Motor position and velocity vary with encoder position and velocity.

For example, an encoder might be attached to a turn-knob on a control panel, so that turning the knob causes a stepper motor to turn.

Setting up encoder following mode

Set up encoder on shaft with knob or other desired mechanism.

6. Connect the encoder to the board. Refer to Figure 6 Feedback Encoder Connectors, page 57.
7. Turn the shaft to its desired starting position.
8. Move the motor to its desired starting position using move commands such as *P* or *D,* addressed to the axis, e.g., */1aM2D100000R*.
9. Once the shaft and the motor are at the desired starting positions, zero the motor and encoder using the *z0* command addressed to the axis, e.g., */1aM2z0R*.
10. Turn on the encoder following mode by issuing *n64* for Axis 1 or Axis 2, whichever is appropriate. For example, */1aM2n64R* turns the mode on for Axis 2 on Drive 1.
11. Turn the mechanism to which the encoder is attached and observe how the motor moves.

    Example command string (after motor has been moved to desired starting position): */2aM2z0am256n64R*

    TBD does z0 zero the following encoder

Commands for encoder following mode

n64 Enters encoder following mode.

z0 Zeroes motor with encoder. Sets encoder count to zero at the current motor and encoder positions. Both the motor and the encoder are to be turned to their desired zero positions before this command is issued. The motor will turn in different

directions above and below the zero point, unless it is set at a physical extreme.

am Multiplier. Sets the ratio of motor movement to encoder movement. If set to 256, the ratio is 1:1; if set to 512 the ratio is 2:1, etc. Default setting is 256.

Encoder position correction mode (*n8*)

Overview

Position correction mode moves the motor until the quadrature encoder count (not microsteps) correctly represents the commanded position. If the motor stalls during a move, the move will be re-attempted until the encoder reads the correct number. If the move cannot be completed, an error message is issued.

There are two main types of encoder feedback arrangement: the first is where the encoder is placed on the motor shaft. The second is where the encoder is placed on the component that is finally positioned, and may be only loosely coupled to the motor due to backlash, etc. The position correction mode will work with either feedback arrangement.

Functional description

In position correction mode, the EZStepper® determines a stalled or overload condition by checking to see if the encoder is tracking the commanded trajectory. If the encoder is not following the commanded trajectory within the error specified by *aC*, a stall is determined. The drive will automatically retry any stalled moves up to the number of times set by the *au* command. If the motor remains stalled, an overload condition is determined. The EZ4AXIS will NOT halt any strings or loops upon detection of a stall.

When an overload condition is detected, it is reported back as an upper or lower case "*I*" (Error 9). Status messages are explained in Appendix 4. This status can be used by an external computer to execute a recovery script (or alternatively, the drive can be configured to execute stored recovery scripts as described in "Auto recovery in position correction mode *(n512, n1024, n1536)*," page 65.)

The position correction mode utilizes a correction algorithm, which operates to bring the motors to within the deadband value set by the *aC* command. This algorithm runs continuously while the stepper motor is in motion and will detect a motor that is stalled or lagging by more than *aC* during a move. At that point the axis will stop and reissue the move starting from zero velocity so as to slowly spin up a motor that may have stalled at high speed. It is important to set the *aC* value to a reasonable minimum number (for example, *50,* the default value) or the drive will always be attempting to correct.

Subsequent disturbances greater than *aC* will result in the correction algorithm being engaged and busy status being asserted.

Commands for encoder position correction mode

n8 Enters position correction mode.

aE Sets encoder ratio, the ratio between encoder movement and stepper motor movement. Default is 1000. Range is 1000-10^6.

NOTE: If the encoder ratio (*aE* command) has been set, the units of the move and velocity commands change to microstep-equivalent encoder counts/second.

au (optional) Sets overload timeout, the number of times that a move will be retried in case of a stall before an overload error notification is sent. Default is 10. Range is 1-64999.

aC (optional) Sets position correction deadband, the distance (in encoder counts) that the stepper is allowed to be in error before position correction using encoder feedback is applied. Default is 50 encoder counts. Range is 1-64999.

?8 Reports encoder position (count) on one axis.

?a8 Reports encoder position (count) on all axes

NOTE: It is sometimes required to set hold current (the *h* command) at least equal to move current (the *m* command) so that there is no reduction in torque at the end of a move. Lower holding torque allows the detent torque to influence the position of the motor.

Setting Up encoder position correction mode

For this procedure, the encoder must be mounted in the position in which it will be used, and wired to the EZ4AXIS17XR.

1. Make sure motor turns in the positive in the direction you intend

First switch the A+ and A- on one phase coil of the motor until the motor turns positive in the direction intended for the mechanism you have. Confirm that the encoder count increases when the motor moves in the positive direction using the *?a8* command, e.g., */1?a8*. If not, swap the AB lines from the encoder, which will reverse the count direction.

2. Calculate encoder ratio and set (*aE*)

Do this if the encoder is on the motor shaft, or if you know that the motor and encoder are coupled so that they turn at exactly the same speed. Otherwise skip to the next step, “Automatically compute encoder ratio and set (aE).”

NOTES

- If the encoder ratio is changed from its default of 1000, the allowed maximum position will be decreased from +2^31 by the

same ratio as the change. The count will roll over from positive to negative range when this limit is exceeded.

- Once the encoder ratio has been set, the units of the move and velocity commands change from microsteps to microstep-equivalent encoder counts/second.

Procedure:

1. Move the motor to zero (starting) position. Typically this is the home position.
2. Calculate the encoder ratio:

   Encoder ratio = (motor microsteps per rev/quadrature encoder counts per rev) X 1000.

   For example, a motor with 200 steps/rev operating on the EZ4AXIS17XR, in 1/16 microstep mode, would have 200 X 16 = 3200 microsteps/rev.

   A 400-line quadrature encoder would have 400 X 4 = 1600 encoder counts per rev.

   So, (3200/1600) X 1000 = 2000 for the encoder ratio, if the encoder is mounted on the motor shaft.
3. Issue the encoder ratio command *aE* with the calculated ratio, e.g., in the preceding example, */1aM2aE2000R* Axis 2.

   NOTE: Preferably, the calculated ratio will be a whole number. If it is not, use the nearest whole number. Later, the *aC* value will need to be adjusted as discussed below.
4. Confirm the calculated encoder ratio by running the next step and comparing it with the automatically computed encoder ratio. Afterwards it will be necessary to re-enter the ratio using the *aE* command.

3. Automatically compute encoder ratio and set (*aE*)

Use this method when the encoder is not placed on the motor shaft and the movement ratio of the shaft to the encoder is unknown. Also use it to confirm an encoder ratio that you have calculated, as above.

NOTE: If the encoder ratio is changed from its default of 1000, the allowed maximum position will be decreased from +2^31 by the same ratio as the change. The count will roll over from positive to negative range when this limit is exceeded.

Procedure:

1. Issue the *n0* command to the axis. This clears any special modes, e.g., */1aM2n0R*.
2. Move the motor to zero (starting) position. Typically this is the home position.
3. Issue the *z0* command to the axis. This zeroes both the motor and the encoder positions.

4. Move the motor 100,000 microsteps by issuing the command *A100000* to the axis. Confirm that the move completes at a velocity that does not stall.
5. If the motor stalls, the velocity will probably need to be reduced using the *V* command. The current programmed velocity can be read by issuing *?2* to the axis.
6. Issue the command *?0* to the axis, which reads back the current position of the motor. If it has not stalled, the number will be 100000.
7. Issue the command *?8* to the axis, which reads back the encoder position (count).
8. Issue the command *aE0* to the axis, which divides these two numbers to arrive at the encoder ratio.
9. Issue the command *?aE* to the axis, which reads back the computed encoder ratio.
10. The value read back is not the precise ratio, but is a very close approximation with an error of a few counts. The actual ratio will be apparent to someone familiar with typical encoder ratios (for example, if the number is read back value is 2557, the actual ratio would be 2560). Use the actual ratio in the next step.
11. Or, if using this procedure to confirm a ratio calculated in the preceding instructions, compare readback with calculated value; they should be approximately the same.
12. Issue the command *aE* with the actual ratio to the axis, e.g., */1aM2aE2560R* from the example above. This overwrites the automatically computed ratio set with the *aE0* command.

NOTE: Overwriting is a crucial step in preventing accumulative error and "shuffling" that would result as the error was repeatedly corrected.

4. Optional: set correction deadband (*aC*)

This value represents the error in quadrature encoder counts allowed before a correction is issued. Default is 50.

- Recommend leaving at default unless operation requires changing. Refer to “Troubleshooting” on page 64 and the “Functional description,” above.
- If calculated ratio is not an even number, set *aC* to cover at least the difference between the calculated ratio and nearest full number.
- Example: */1aM2aC75R*

5. Optional: Set the Overload Timeout Value (*au*)

This is the number of retries allowed under a stall condition. Once this number is reached and the motor remains stalled, the motor is in an overload state and will turn off encoder feedback. Default is 10.

Example: */1aM2au10000R*

6. Enable the position correction mode (*n8*)

1. Zero positions just prior to enabling position correction mode by issuing the *z0* command to the axis, with the motor in the zero position (typically home).

   Example: */1aM2z0R*

2. Enable position correction mode by issuing the *n8* command to the axis, e.g., */1aM2n8R*

   Example of full command string:

   /1aM2z0aC50h40m40au100aE2000L100n8R

   This command string starts the position correction mode on Axis 2 on Drive 1 and sets parameters discussed above including, in addition, hold current, move current, and acceleration (*h*, *m,* and *L*).

   Remember that the motor should be moved to its zero position (typically the home position) before issuing the command string, because it includes the zero command (*z*).

To terminate position correction mode

To terminate, issue the following (Axis 2 used for example):

/1aM2n0R Exits current mode.

/1T2 Terminates any commands in process on the axis (Axis 2).

Troubleshooting

- If motor consistently stops during a move:

  If a very fine line count encoder is used, such that for example, the encoder ratio is around 2000, or if the encoder is decoupled from the motor shaft, or if the encoder ratio is non integer, increase the error (*aC*) allowed, for example, set *aC* to *2000*. This way a move that is in progress will not be halted and restarted because the correction algorithm detects that the position error is too large.

- Typical reasons that the position error is too large:
  - Incorrect or non integer *aE* value.
  - *m* value (move current) set too low.
  - *L* value (acceleration) set too high for torque available from motor. Use *?L* to read back the current acceleration setting for the axis, e.g., */1?L*.
  - *V* value (velocity) set too high for torque available from motor. Use *?aV* to read back the current velocity setting for all axes, e.g., */1?aV* (requires v7.50 and higher firmware). Results are displayed in order of axis numbers, e.g., 1000,50000,1000,10000 lists the velocity settings for axes 1 through 4 respectively.
  - Physical obstruction or excessive friction
  - Inadequate voltage from power source

Auto recovery in position correction mode (*n512*, *n1024*, *n1536*)

NOTE: This function is not implemented yet

Overview

In the Position Correction mode, an error message (upper or lower case *I* —Error 9) is issued if a stalled motor cannot be corrected within the number of retries set by the *au* command. This message indicates an overload status, and can be used by an external computer to execute a recovery script. For information regarding status messages, see "Appendix 4, Device Response Packet," page 89.

However, it may be desired that the drive recover by itself in the case of a stand-alone application. The *n512*, *n1024*, and *n1536* auto recovery modes allow the EZ4AXIS to execute stored recovery scripts when an overload is detected.

These scripts, provided by the user, are stored in EEPROM locations 13, 14, and/or 15. In addition, a last-resort script is stored in location 12.

Depending on which auto recovery mode is selected, the drive will execute stored program 13, 14, or 15 when an overload is detected. The number of times the recovery script can run is set by the *u* command (see below). If the selected recovery script cannot auto recover within the number of retries specified by the *u* command, program 12 is run.

An overload error on any motor, if position correction is enabled, will execute error recovery.

Commands

n512 Enters Auto Recovery mode and designates the program stored in location 13 as the recovery script.

n1024 Enters Auto Recovery mode and designates the program stored in location 14 as the recovery script.

n1536 Enters Auto Recovery mode and designates the program stored in location 15 as the recovery script.

When issuing the commands above, it is necessary to combine them with the position correction mode command *n8*. For example, *n512+n8 = n520*, *n1024 + n8 = n1032*, or *n1536 +n8 = n1544*. This includes position correction with auto recovery. Otherwise position correction will be disabled when auto recovery is enabled.

u Sets number of times the recovery script may run before executing the last-resort recovery script stored in location 12 , e.g., *u5* = 5 times.

Example: */1aM2u5n1032R*. This sets up auto recovery on Drive 1 Axis 2, using Program 14 for the initial recovery script (*n1032*, which combines *n1024* with *n8*) to run up to 5 times (*u5*).

The user will need to provide the recovery scripts and store them in program locations 13, 14, and/or 15, and 12.

Example (exercise)

(Axis 2 is used in the example command string):

1. Enter */1s13p1202R*. This stores an error recovery script in memory location 13 that simply sends the number 1202 to the bus, each time the recovery script is run.
2. Enter */1s12p1201R*. This stores a recovery script in memory location 12, which simply sends the number 1201 to the bus. This is the final "last resort" script.
3. Move the motor to its starting position (usually the home position).
4. Enter */1aM2m5h5z0aC50au5u3aE2000L100n520R*.

   This does the following:

m5 and *h5*	Sets move current and hold current low so the motor can be stalled easily.
z0	Zeroes encoder and motor position counter with the *z0* command.
aC50	Sets correction deadband at 50 encoder counts.
au5	Sets retries allowed via encoder feedback to a maximum of 5.
u3	Specifies that recovery script is run a maximum of three times.
aE2000	Sets encoder ratio to 2000 (encoder ratio setup is described in "Calculate encoder ratio and set (aE)" on page 61.
n520	Enters position correction mode and specifies that stored program 13 will be executed on overload error condition. *n* = 520 = *n8* + *n512*.

5. Now manually move the motor shaft so that the drive tries to correct five times and then gives up.

   After the fifth retry, the overload error will be issued and the drive will execute stored program 13 up to three times, sending the number 1202 to the bus each time. If the motor is manually held so that it remains stalled, the number 1202 will be sent 3 times followed by the number 1201 as the final recovery script, stored program 12, is run.

Encoder Overload Report mode (*n16*) –not implemented yet

In position correction mode (*n8*), the drive will automatically correct any stalled moves up to the limit given by "*au*." Only then will it report Error 9, the overload error indication.

However, it may be desirable to detect a stall but not correct it. The Overload Report Mode does just this. The encoder value is continuously compared against the commanded position and Error 9 is set when these do not match to within the error band specified by the *aC* command (default 50 encoder counts). When this error occurs, the axis will exit from any loops or multiple command strings it may be executing.

This mode requires the encoder ratio to be entered correctly via the *aE* command. Enter encoder ratio and zero the encoder and motor position counter prior to issuing the *n16* command (see instructions below).

Overload Report mode commands

z0 Zero encoder and motor together. First move motor to zero position (typically home).

aE Encoder ratio. Set according to instructions in "Calculate encoder ratio and set (aE)" on page 61.

aC Specifies correction deadband. Default is 50.

n16 Enters overload report mode.

Example command string:

/1aM2z0aE2000n16R—This sets up Overload Report Mode on Axis 2 on Drive 1. Before issuing this command string, send the motor to the zero position (typically home).

Setting arbitrary measurement units via the *aE* command

It may be desired to use inches, for example, as measuring units in a particular setup that controls the depth of a drilling apparatus.

If the motor movement vs. drill depth is known, this information can be used to set the encoder ratio to a value that allows movement in units that reflect the desired precision, e.g., thousandths of an inch.

It is not necessary to use position correction mode (*n8*) or even have an encoder, in order to set the encoder ratio */1aE32000R* etc. Setting the encoder ratio thus allows positioning in any units of the user's choice.

Appendix 1. Addressing methods reference

Addressing individual drive cards

NOTE: The following addresses correspond to the settings of the address switches on the individual drive cards (Not to be confused with different axes on the same card).

Addressing drives 1-9

Use */1*, */2*, etc. with address switch on board set accordingly.

Addressing drives 10-16

Use the ASCII characters on a standard keyboard:

Bus address	Type this:	Set address switch to:
10	*/:* (colon)	A
11	*/;* (semi colon)	B
12	/< (less than)	C
13	/= (equals)	D
14	/> (greater than)	E
15	*/?* (question mark)	F
16	/@	0 (zero)

(So these addresses 1 to 16 map to hex 30 to hex 3F on the ASCII chart)

Addressing one axis (motor) within a single drive card

NOTE: If no axis is selected, any command issued is addressed to the default axis. Any multi-axis command will reset the default axis to Axis 1. Otherwise the default axis is the last axis addressed in a command.

NOTE: Use the *aM* command to address the axes, *aM1* through *aM4* for axes 1 through 4.

Select axis and issue command at the same time

Example: */1aM1A1000A0R* for Axis 1 (*aM1*)

Pre-select axis and send commands subsequently

Once the axis is selected, subsequent single-axis commands and queries are directed to that axis until another axis is selected or a multi-axis command is issued.

Example:

/1aM4R	Selects Axis 4, then:
/1A1000R	Moves Axis 4 to absolute position 1000
/1?0	Retrieves Axis 4 position
/1L10R	Sets Axis 4 acceleration

Addressing multiple axes on a drive card simultaneously

This function is available with multi-axis and interpolation commands. Please refer to “Send commands to multiple axes (multi-axis commands)” on page 17 and “Drawing circles and lines” on page 98 (Appendix 9).

Addressing banks of drive cards

Up to 16 drive cards can be addressed in banks of 2, 4, or "all," increasing versatility and ease of programming. These are physically separate cards, not to be confused with different motors on the same drive.

Addressing banks of two drives

Drives 1 and 2	*/A*
Drives 3 and 4	*/C*
Drives 5 and 6	*/E*
Drives 7 and 8	*/G*
Drives 9 and 10	*/I*
Drives 11 and 12	*/K*
Drives 13 and 14	*/M*
Drives 15 and 16	*/O*

Addressing banks of four drives

Drives 1, 2, 3, and 4:	*/Q*	
Drives 5, 6, 7, and 8:	*/U*	
Drives 9, 10, 11, and 12:	*/Y*	
Drives 13, 14, 15 and 16:	*/]*	(close square bracket)

Addressing all drive cards at once

Use the global address /_ (underscore) to select all drives. To address all axes on the drives, insert the command four times separated by commas.

To select all drives on bus: /_

Appendix 2. Command set reference

Introduction

The following table lists the commands available for the EZ4AXIS at the time of publication. It indicates the command, possible operands, and brief descriptions with examples.

NOTE: The EZ4AXIS is always in 1/16th step mode, meaning that there are always 16 microsteps per step.

Command list

Table 1. Command Set

Command (case sensitive)	Operand/ (default)	Description
	AXIS SELECTION	
aM	1, 2, 3, or 4 (1)	**Designate target axis for command.** */1aM1R* selects Axis 1. From then on, all commands are sent to Axis 1. */1aM2R* selects Axis 2, and so on. As part of a complete move command: e.g., */1aM2A1000R*. Move Axis 2 to absolute position 1000.
	POSITIONING	
A	0-2^31	**Move motor to absolute position.** (microsteps or quadrature encoder counts, 32-bit positioning). E.g. */1aM3A10000R* (Axis 3 specified) Or Multi Axis */1A1000,2000,3000,1234R*
P	0-2^31	**Move motor relative in positive direction.** (microsteps or quadrature encoder counts) E.g. */1aM2P10000R* (Axis 2 specified) A value of zero will cause an endless forward move at speed V. (i.e., enters into velocity mode) The velocity can then be changed on the fly by using the *V* command. Endless moves can be terminated by issuing a *T* command or by a falling edge on the Switch 2 Input. See *T* command. Or Multi Axis */1P1000,2000,3000,1234R*

Command (case sensitive)	Operand/ (default)	Description
D	0-2^31	**Move motor relative in negative direction.** (microsteps or quadrature encoder counts) E.g. */1aM2D10000R* (Axis 2 specified) A value of zero for the operand will cause an endless backwards move at speed *V*. (i.e. enter into Velocity Mode). The velocity can then be changed on the fly by using the *V* command. Endless moves can be terminated by issuing a *T* command or by a switch if "kill move" is enabled Or Multi Axis */1D1000,2000,3000,1234R* Or Multi Axis */1D1000,,,1234R to move axes 1,4*
F	0 or 1	**Set direction of rotation considered positive to default or specified axis.** E.g. */1F1R or /1aM2F1* with axis (Axis 2) specified Only do this once on power up, do not use with encoder feedback). Swapping motor wires also works. Or Multi Axis */1F1,1,0,1R*
j	1 or 2 or 16 or 32	**Set Microstep resolution** E.g. */1j16R* will set the microstep resolution to be 1/16th microstep. Note that All motion units are in microsteps
r	NA	**Set current position to be same as encoder position.** E.g. */1aM2rR* (Axis 2 specified)
INTERPOLATION COMMANDS (Axes 1, 2 and separately Axes3,4)		
aaA	0-90000	**Set diameter of circle. (circular interpolation)** Units are micro6steps. - Not Implemented yet
aaI	0-1024	**Set beginning phase of circle (circular interpolation).** - Not Implemented yet Units are such that 360 degrees = 1024.
aaW	1-20000	**Set speed at which circle is drawn (circular interpolation).** - Not Implemented yet Units are microsteps/second.
aaC	1-1024	**Specifies how much of a circle is drawn and initiates circle drawing process (circular interpolation).** Units are such that 100% = 1024 = 360°. Note: *aaC2048* will draw two circles on top of one another. - Not Implemented yet

Command (case sensitive)	Operand/ (default)	Description
an65536	NA	**Places axes 1 and 2 and separately axes 3 and 4 interpolation mode.** This mode draws straight lines. In this mode, the speed of the faster axis is slowed such that both axes arrive at the destination at the same time. You may issue a multi-axis command addressed to all four axes to enter linear interpolation mode. In such a case Axes 1 and 2 will move in interpolation fashion, and separately Axes 3 and 4 can move in a linear interpolated fashion. *1an65536,65536,0,0R sets* Axes 1 and 2 into linear interpolation mode for drawing straight lines. */1an0,0,65536,65536R sets* Axes 3 and4 into linear interpolation mode for drawing straight lines. */1an65536,65536,65536,65536R sets* Axes 1 and 2 into linear interpolation mode and separately sets axes 3 and 4 into interpolation mode
	HOMING	
f	0 or 1 (0)	**Set Home Flag and Limit polarity,** Sets polarity of limits/home sensor. Each axis has own set of limit/home flags; these are on the 6-pin connectors. The polarity of each limit/home flag can be changed individually. E.g. */1aM3f1R* sets Axis 3 limit/home flag polarity to 1. 1=high (normally closed) 0=low (normally open)
Z (upper case)	0-(2^31) (400)	**Home/initialize motor.** Initializes motor to known position. When issued, motor will turn toward 0 until the home sensor is interrupted. If already interrupted, motor will back out from interrupted position and come back in until re-interrupted. This sets motor position to zero. Includes number of microsteps allowed before reaching home position. E.g. */1aM3 Z300000R* (Axis 3 specified, 300000 microsteps allowed)
z0 (lower case)	--	**1. When used with encoder: *z0* zeroes motor with encoder at current position.** **2. In voltage positioning (potentiometer positioning), voltage-velocity (joystick), and position correction modes: sets zero point to current motor position.** E.g., */1aM2z0R* sets zero point to current motor position. Absolute positions are computed in reference to this point.

Command (case sensitive)	Operand/ (default)	Description
		SET VELOCITY
V	1-59900 (568)	**Set max/slew speed (velocity) of default or selected motor. (positioning mode)** Sets microsteps per second. Max V is 59900. NOTE: The EZ4AXIS is always in 1/16th step mode. E.g. */1aM2V10000R* sets max. velocity of Axis 2 motor. Or send /1V1000,1000,1000,1000R NOTE: If the encoder ratio (*aE* command) has been set, the units of velocity change to encoder counts/second.
v	0-900 (0)	**Set start velocity for selected motor.** Units are microsteps/second. Only or applications where it is desired to have motor accelerate suddenly from zero to a specific velocity. Else do not use, causes jerky motion. E.g. */1aM2v100R* sets start velocity of Axis 2 motor to 100 microsteps/second.
c	0-900 (0)	**Set stop velocity for selected motor.** Units are microsteps/second. **Note set c<v** Only for applications where it is desired to have motor stop suddenly from a specified velocity rather than follow the more sloping deceleration curve. Else do not use, causes jerky motion. E.g. */1aM2c400R* sets stop velocity of Axis 2 motor to 400 microsteps/second.
		SET ACCELERATION
L	0-64999 (10)	**Set acceleration factor.** Acceleration factor = *L** (*V* in microsteps / sec^2) = (*L* value) x (TBD). For example, if *V*=10000 and *L*=1, it will require TBD seconds to reach final velocity. **NOTE:** Acceleration does not scale with encoder ratio. E.g., */1aM21LR* (Axis 2 specified)
		LOOPING AND BRANCHING
g	NA	**Beginning of loop marker.** E.g. */1aM2gP10000M1000G10R*. (Axis 2 specified) This loop begins with the *P* command following the *g* marker. It continues until the *G* marker (described below).
G	0-30000	**End of loop marker and repeat designator.** G marks the end of a loop. The operand following the G specifies how many times to repeat the loop. A value of 0 causes the loop to repeat until terminated. (Requires *T* command to terminate). If no value is specified, 0 is assumed. E.g. */1aM2gP10000M1000G10R*. This loop repeats 10 times (shows Axis 2 specified). NOTE: Loops can be nested up to 4 levels. E.g. */1aM2gA1000A10000gA1000A10000G10G100R* is an example of a two-level nested loop.

Command (case sensitive)	Operand/ (default)	Description
H	101-414	**Halt current command string and wait until input or limit switch condition specified.** 101wait for low on Axis 1 ADC1 111wait for high on Axis 1 ADC1 102wait for low on Axis 1 ADC2 112wait for high on Axis 1 ADC2 103wait for low on Axis 1 limit 1 (lower) 113wait for high on Axis 1 limit 1 (lower) 104wait for low on Axis 1 limit 2 (upper) 114wait for high on Axis 1 limit 2 (upper) 201wait for low on Axis 1 ADC1 211wait for high on Axis 1 ADC1 202wait for low on Axis 1 ADC2 212wait for high on Axis 1 ADC2 203wait for low on Axis 1 limit 1 (lower) 213wait for high on Axis 1 limit 1 (lower) 204wait for low on Axis 1 limit 2 (upper) 214wait for high on Axis 1 limit 2 (upper) 301wait for low on Axis 1 ADC1 311wait for high on Axis 1 ADC1 302wait for low on Axis 1 ADC2 312wait for high on Axis 1 ADC2 303wait for low on Axis 1 limit 1 (lower) 313wait for high on Axis 1 limit 1 (lower) 304wait for low on Axis 1 limit 2 (upper) 314wait for high on Axis 1 limit 2 (upper) 401wait for low on Axis 1 ADC1 411wait for high on Axis 1 ADC1 402wait for low on Axis 1 ADC2 412wait for high on Axis 1 ADC2 403wait for low on Axis 1 limit 1 (lower) 413wait for high on Axis 1 limit 1 (lower) 404wait for low on Axis 1 limit 2 (upper) 414wait for high on Axis 1 limit 2 (upper) E.g. */1gH202P10000G20R*. Does the loop when Axis 2 Input 2 is low. Also H202H212 will look for an edge. If halted, operation can also be resumed with the *R* command, e.g., */1R*.

Command (case sensitive)	Operand/ (default)	Description
S	101-414	**Skip next instruction depending on input status.** 101wait for low on Axis 1 ADC1 111wait for high on Axis 1 ADC1 102wait for low on Axis 1 ADC2 112wait for high on Axis 1 ADC2 103wait for low on Axis 1 limit 1 (lower) 113wait for high on Axis 1 limit 1 (lower) 104wait for low on Axis 1 limit 2 (upper) 114wait for high on Axis 1 limit 2 (upper) 201wait for low on Axis 1 ADC1 211wait for high on Axis 1 ADC1 202wait for low on Axis 1 ADC2 212wait for high on Axis 1 ADC2 203wait for low on Axis 1 limit 1 (lower) 213wait for high on Axis 1 limit 1 (lower) 204wait for low on Axis 1 limit 2 (upper) 214wait for high on Axis 1 limit 2 (upper) 301wait for low on Axis 1 ADC1 311wait for high on Axis 1 ADC1 302wait for low on Axis 1 ADC2 312wait for high on Axis 1 ADC2 303wait for low on Axis 1 limit 1 (lower) 313wait for high on Axis 1 limit 1 (lower) 304wait for low on Axis 1 limit 2 (upper) 314wait for high on Axis 1 limit 2 (upper) 401wait for low on Axis 1 ADC1 411wait for high on Axis 1 ADC1 402wait for low on Axis 1 ADC2 412wait for high on Axis 1 ADC2 403wait for low on Axis 1 limit 1 (lower) 413wait for high on Axis 1 limit 1 (lower) 404wait for low on Axis 1 limit 2 (upper) 414wait for high on Axis 1 limit 2 (upper) E.g. */1gS202P10000GR*. Only does the P10000 when Axis 2 Input 2 is high.

Command (case sensitive)	Operand/ (default)	Description
PROGRAM STORAGE AND RECALL		
e	0-15	**Executes program stored in specified EEPROM location 0-63.** E.g. */1e1R* executes stored program 1 (the program stored in EEPROM location 1).
s	0-15	**Stores a program to specified EEPROM location 0-63.** E.g. */1s1A10000A0R* stores command string to location 1. This command takes approx 1 second to write to EEPROM. NOTES: 25 full commands max. per string, 256 characters. Program 0 is executed on power-up. If no command string is included, content of memory location is erased, eg */1s14R* erases program 14
PROGRAM EXECUTION		
R	NA	**Run the command string that is currently in the execution buffer of the default or selected motor.** E.g. */1R* or */1aM2R* (Axis 2 specified) Can be used to resume operation after a halt (*H*) or termination (*T*). Resume causes the last command issued to be run.
SET MAX MOVE / HOLD CURRENT		
h	0-50 (10)	**Sets hold current within a scale of 0 to 50% of max current.** 100% = 0.5A E.g. */1aM2h15R = 15%* (Axis 2 specified). Applies to specified axis.
m	0-100 (25)	**Set max move current within a scale of 0 to 100% of max current.** 100% = 0.5A E.g. */1aM2m40R = 40%* (Axis 2 specified) Applies to specified axis.
N MODE COMMANDS		
N	1-3 (1)	**Initiates designated mode determined by operand.** *1* = Encoder with no index or no encoder (default). Homes to opto or switch. *2* = Encoder with index. Homes to index. *3* = Uses potentiometer as an encoder on specified axis. E.g., */1aM1N3* Applies to specific axis.

Command (case sensitive)	Operand/ (default)	Description
n MODE COMMANDS		
n	0-128000 (0)	**Initiates designated mode determined by operand.** Bit0 (LSB) - */1n1R* – Not implemented yet Bit1 - */1aM1n2R* Enable limits on an axis-by-axis basis. Limits are enabled for Axis 1 in this example. The polarity of the limits is set by the *f* command. Bit2 - */1n4R* – Not implemented yet Bit3 - */1n8R* enables Encoder Position Correction mode, with the two encoder (AB) inputs being used for feedback. Bit4 - */1n16R* Enables Encoder Overload Report mode. Bit5 - */1n32R* – Enable Encoder AB mode or Step and Dir mode for auxiliary encoders 5 and 6. Bit8 - */1n64R* Enable Encoder following mode for auxiliary encoders 5 and 6. Also use in combination with *n32* mode eg *n96* will follow a step and dir input. Bit7 - */1n128R* – Not implemented yet Bit8 - */1n256R* – Not implemented yet Bit9 and Bit10 - These bits will execute one of the stored recovery script programs 13, 14 or 15 whenever the position correction feedback shuts down the drive due to an overload. (That is, the number of retries specified by the au command has been exhausted. See Position Correction Commands in this table.) Axes 1 and 2 only. Position Correction must run concurrently. (Requires firmware version 6.997 or higher.) */1n512R* will execute recovery program 13. */1n1024R* will execute recovery program 14. */1n1536R* will execute recovery program 15. Bit11 - */1n2048R* – Not implemented yet Bit12 - */1n4096R* – Not implemented yet Bit13 - */1n8192R* – Not implemented yet Bit14 - Reserved Bit15 - Reserved Bit16 - */1n65536R* Enables joystick (voltage-velocity) mode, in which a potentiometer or other varying voltage between 0 and 3.3V can be used to control the velocity of Axis 1. **NOTE: Any n mode change may require up to 20mS to propagate through the firmware. If modes are being dynamically changed a small wait command such as M20 may be required before a move command that follows it.**

Command (case sensitive)	Operand/ (default)	Description
an MODE COMMANDS		
an	0 or 65536	**Places axes 1,2 into linear interpolation mode and/or place axes 3,4 into linear interpolation mode.** Bit0 (LSB) – Not implemented yet Bit1 - */1aM1n2R* Enable limits on an axis-by-axis basis. Limits are enabled for Axis 1 in this example. The polarity of the limits is set by the *f* command. Bit2 - */1n4R* – Not implemented yet Bit3 - */1n8R* enables Encoder Position Correction mode, with the two encoder (AB) inputs being used for feedback. Bit4 - */1n16R* Enables Encoder Overload Report mode. Bit5 - */1n32R* – Not implemented yet Bit6 - */1n64R* Enable Encoder Following mode. Axes 1 and 2 only. Bit7 - */1n128R* – Not implemented yet Bit8 - */1n256R* – Not implemented yet This mode draws straight lines. In this mode, the speed of the faster axis is slowed such that both axes arrive at the destination at the same time. E.g., */1an65536,65536,0,0A1000,2000,100,100R* */1an65536,65536,65536,65536A1000,2000,100,100R* */1an0,0,65536,65536A100,200,1000,2000R*
POSITION CORRECTION COMMANDS – Not Impemented yet		
aC	1-64999 (50)	**Set position correction value (deadband).** When in position correction mode, sets distance (in quadrature encoder counts) from commanded position allowed before the drive corrects using encoder feedback. E.g. */1aM2aC100R* (Axis 2 specified)
aE	1000-10^6 (1000)	**Set encoder ratio.** This sets the ratio between the encoder counts/rev and the microsteps/rev for the specified motor. E.g. */1aM2aE12500R* (Axis 2 specified) Encoder ratio = (motor microsteps per rev/quadrature encoder counts per rev) X 1000.
au	1-64999 (10)	**Set overload timeout.** This sets the number of times the move is retried in case a move stalls. E.g. */1aM2au10000R* (Axis 2 specified) When the *au* retries are exhausted, the drive will drop out of position correction mode (*n8*) and report Error 9 (overload). Also see the auto recovery n mode commands *n512*, *n1024*, and *n1536R*.

Command (case sensitive)	Operand/ (default)	Description
u	1-64999 (0)	**Sets the number of times error recovery scripts 13, 14, or 15 are run prior to calling upon final recovery script 12.** (Requires firmware version V6.99 or higher.) Also see the auto recovery n mode commands *n512*, *n1024*, and *n1536R*.
POWER DRIVER CONTROL		
J	0-15 (0)	**Turn driver On/Off** Digits following the *J* command are interpreted as 4-bit binary equivalents: 1111 binary = 15 decimal = all drivers on /1gJ15,15,15,15J0,0,0,0GR will toggle all outputs on and off /1gJ1,0,0,0J0,0,0,0GR will toggle output 1 on axis 1 on and off tection.
ak aak	250-511	**Changes driver outputs to PWM and sets PWM duty cycle.- Not Implemented yet** *ak* sets Driver #1; *aak* sets Driver #2 e.g., */1ak381R* or */1aak381R* Where 250=full off (0%) and 511=full on (100%).
POTENTIOMETER POSITION COMMANDS		
These commands apply to the Potentiometer Positioning mode (*n8192*). **Not Implemented**		
ad	0-16368 (50)	**Sets a deadband (in microsteps) around the potentiometer value used for the last move.** This deadband must be exceeded before a new move command is issued. The deadband is defined in terms of the reading from the potentiometer after A/D conversion, a number ranging from 0–16368 which represents 0–3.3V. E.g. */1aM2ad100R* (Axis 2 specified)
am	0-20000 (256)	**Set A/D multiplier.** The potentiometer output value is multiplied by this value and divided by 256 to get the preliminary commanded position. E.g. */1aM2am512R* (Axis 2 specified).
ao	0-20000 (0)	**Specify positioning offset.** After multiplication by the *am* value, this offset is added to obtain the final commanded position. E.g. */1aM2ao1228R* (Axis 2 specified)
MISCELLANEOUS COMMANDS		
aP	0-30000 (5)	**Response delay** (Available in V6.79+) Units are milliseconds. This command sets the delay from the drive receiving the command to the response being sent out. E.g. */1aP1000R* sets the delay to 1000 milliseconds.

Command (case sensitive)	Operand/ (default)	Description
ap	0-15	**Not implemented** **Inverts polarity of inputs on 8-pin connector as seen by *?4*, *S*, and *H* commands.** (Available in firmware version 7.60G4+) Example: */1ap3R* will invert the polarity of inputs 1 and 2. Value is decimal number seen as combination of 4 binary bits.
b	9600 19200 38400 to 230400 (9600)	**Adjust baud rate** E.g. */1b19200R* This command will usually be stored as program zero and executes on power-up. Default baud rate is 9600. NOTE: correct termination and strict daisy chaining required for reliable operation at higher baud rates.
K	0-64999 (0)	**Set backlash compensation.** Units are number of steps. For when a non-zero value of K is specified, the drive will always approach the final position from a direction going more negative. If going more positive, the drive will overshoot by an amount K and then go back. By always approaching from the same direction, the positioning will be more repeatable.
M	0-29999	**Wait for specified period.** Units are milliseconds.
p (lower case)	0-64999	**Ping Command** Sends a numeric message back to the host, when that point in the command string is reached. E.g. */1aM2gA1000p3333A0G0R* (Axis 2 specified). Will send the number 3333 every time through the loop. Example: /0@3333ÿ/0@3333ÿ/0@3333 *No Error* **Note:** Care must be taken when using this command because it can tie up the 485 bus.
ANALOG INPUT (ADC) COMMANDS		
The Analog/Digital IO and Limit/Home inputs are all ADC.		

Command (case sensitive)	Operand/ (default)	Description
/1**at**	**X100000 to X165535** **X200000 to X265535** **Where X =1-4 depending on axis**	**Sets thresholds for "one" and "zero" on each IO connector** The number consists of the input number followed by a 5-digit number ranging from 00000-65535, which represents the threshold on a scale from 0-3.3V. The default value is 24200 (1.22V) for all four inputs. Note: threshold value= (threshold voltage/3.3) x 65535 Changing the threshold allows the H (Halt) and S (Skip) commands to work on a variable analog input value which essentially allows the program to act upon an analog level. This can be used, for example, to regulate pressure to a given level by turning a motor on/off at a given voltage. IO connector thresholds can be read back with the *?at* command. See the *?at* command for details. **Setting Limit/Home Connector thresholds** Simply add the axis number and a specific limit to the beginning of the command described above. For example, to set Axis 4 upper limit to 10000, issue */1at4210000R* (note seven numerical digits). The numeral 4 specifies Axis 4, and the numeral 2 specifies the upper limit. Note that on the Limit/Home connectors, Input 1 is the lower limit/home input, and Input 2 is the upper limit. ADc input thresholds can be read back with the *?aat* command. See the *?aat* command for details.

IMMEDIATE QUERIES / COMMANDS

The following are "Immediate" queries and commands, which can execute while other commands are running, allowing on-the-fly programming.

- These commands cannot be cascaded in strings or stored.
- These commands must each be sent individually, and separate from axis selection commands. (Note axis selection must be issued separately prior to query /1aM2R<CR> then /1?2<CR> etc)
- These commands do not require an "R" at the end.
- In later versions of firmware it will be possible to query *some* parameters by using the same letter that set the parameter. E.g., */1?A* returns current position; /1?2 returns slew speed and */1?V* returns instantaneous velocity of a moving motor.
- Some other commands may be executed as immediate commands.

Command (case sensitive)	Operand/ (default)	Description
`$`		**Reports the command string currently executing, or most recently executed on drive.** E.g., */1$* Example response: *P1000P1200P1300P1400 No Error* **Note:** The $ command may not be able to read very long command strings. On older firmware, attempting to read very long command strings may kill board operation. If this happens, power cycle the board.
`&`		**Reports the current firmware revision number and date.** E.g., */1&* Example response: *EZController AllMotion V7.50h5 12-18-12 No Error*
`?0` `(?zero)`		**Reports the current commanded position for last commanded axis.** E.g. */1aM2R /1?0* (e.g., for Axis 2) Example response: *10000 No Error*
`/1?A` `/1?aA`		**Reports positions of all four** E.g., */1?A or /1/aA* Example response (Axes 1 through 4 in order): *102000,35000,35000,5000 No Error*
`/1?0` `(?zero)`		**Reports the current commanded position for last commanded axis.** E.g. */1aM2?0* (e.g., for Axis 2) Example response: *10000 No Error*
`?aV`		**Reports programmed velocities of all four motors (requires V7.50 or higher firmware).** E.g., */1?aV* Example response: *568,568,568,568 No Error*
`?1`		**Reports start speed for default or selected motor.** E.g., */1aM2R /1?1* (Axis 2 specified)
`?2`		**Reports the current Slew/Max speed for Position mode for default or selected motor.** E.g., */1aM2R /1?2* (Axis 2 specified)

`/1?41`
`/1?42`
`/1?43`
`/1?44`
		Digital Input Query. Reports the high/low status of all four Digital/Analog IO inputs
		E.g., */1?4*
		0-15 represents a 4-bit binary pattern:
		Bit 0 = Switch1 (input 1) (Thresholded Analog Input 1)
		Bit 1 = Switch 2 (Input 2) (Thresholded Analog Input 2)
		Bit 2 = Opto 1 (input 3) (Home / LoweLimit Input)
		Bit 3 = Opto 2 (Input 4) (Upper Limit Input)
		Example: 11 = all high except input 3 (bit 2)

Command (case sensitive)	Operand/ (default)	Description
?8		**Reports encoder position for currently selected motor.** *E.g., /1aM1R <enter> then* */1?8* (Motor/encoder 1 specified) */1aM2R <enter> then* */1?8* (Motor/encoder 2 specified) Example response: *1000*
?a8		**Reports encoder position all axes.** *E.g., /1?a8* Example response: *1000,300,2990,4567*
?aa1 ?aa2 ?aa3 ?aa4	1-4	**Reports analog values on all 2 Analog Inputs and 2 Analog Drive parameters per axis** E.g. */1?aa1* Example response: *64300,15, 64300, 64300 No Error* Readback order is inputs 4,3,2,1 These numbers represent the voltage range 0-3.3V available at each input, as expressed by a number from 00000-65535
?aat		**Reports thresholds for all four axes on Limit/Home connections.** /1?aat Read thresholds of all ADCs on the four 10 pin I/O connectors on the drive. The order of readback (in terms of the axis/input identifiers explained above) is *LEFT END* is1ADC2, Axis1ADC1, Axis2ADC2, Axis2ADC1, Axi3ADC2, Axis3ADC1, Axis4ADC2, Axis4ADC1. *RIGHT END* Example readback: *6144,6144,6144,6144,9999,6144,6144,6144 No Error*.
?at		**TBD**
?aE		**Reports encoder ratio for default or selected axis.** e.g., */1aM2R <CR> then /1?aE* <CR>(with Axis 2 specified.
?h		**Not Implemented**
?m		**Not Implemented**
?L		**Reports acceleration for default or selected axis.** e.g., */1aM2R /1?L* (Axis 2 specified)
?2 ?V		**/1?2 Reports programmed velocity (slew rate) for default or selected axis. /1?V reports instantaneous velocity** e.g., */1aM2R /1?V* (Axis 2 specified)

Command (case sensitive)	Operand/ (default)	Description
Q		**Reports current status of EZ4AXIS board (drive).** E.g., */1Q* Reports the Ready/Busy status as well as any error conditions in the status byte of the return string. The return string consists of the start character (/), the master address (0) and the status byte. Bit 5 of the status byte is set when the drive is ready to accept commands. It is cleared when the drive is busy, ie any axis is moving. The least significant four bits of the status byte contain the completion code. List of codes: 0 = No Error 1 = Initialization error 2 = Bad Command 3 = Operand out of range Errors in opcode will be returned immediately, while Errors in operand range will be returned only when the next command is issued. See Appendix 4, Device Response Packet, page 89.
General Immediate Query Syntax		In firmware version 7.02 and above it is possible to query *some* parameters by using the same letter that set the parameter. E.g. */1?A* reports current position; /1?aA reports all 4 motor positions.
T		**Terminate current command or loop for an axis.** */1T1* = terminate Axis 1 */1T3* = terminate Axis 3 */1T2* = terminate Axis 2 */1T4* = terminate Axis 4 NOTE: Do not use /1T2, /1T3 etc. to terminate a loop since the behavior is undefined and may change in the future.

Command (case sensitive)	Operand/ (default)	Description
`Other Immediate Commands`		The following other commands may be issued as immediate (on-the-fly): A L D J V P These commands can be sent while the motors are moving and will change the motion “on the fly”. Typically issue in multi axis format to affect any motor. Eg /1P1000,,1000,R will change target position of axes 1 and 3 while moving. Eg issue /1A10000,10000,10000,10000R then while running issue /1A,,-10000,R reverses motor 3 while moving. Note that these commands must be issued one at a time. Ie no multi command strings. If drive is running a sequence of commands in a loop then this command will affect the currently running command instantly, and then the loop will continue, but with cganged velocity etc in case that was changed. By using multi axis on the fly commands any motor can be started at any time, further by using /1T1 /1T2 etc and motor can be stopped at any time.
`Other Immediate Commands`		The following other commands may be issued as immediate (on-the-fly) . The drive will adapt to the new target while running: A, P, D, V, L
RESPONSE PACKET See Appendix 4, Device Response Packet		

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix 3. Step loss detection using opto

For some applications that operate without encoder feedback, it may be necessary to detect loss of steps due to the mechanism stalling for any reason.

Step loss is easily detected by following these steps:

1. Home the stepper to the opto using the *Z* Command.
2. Move out of the flag a little by issuing, for example, an *A100* command.
3. Figure out the exact step on which the flag gets cut by issuing *D1* commands followed by *?4* commands to read back the opto. Let's call this value Y. (This only needs to be done once during initial set-up).
4. Execute the move sequence for which step loss detection is needed.
5. Issue a command to go back to absolute position Y+1.
6. Check the opto; it should not be cut (read opto back with the *?4* command).
7. Now issue a command to go to position Y-1.
8. Check the opto; it should be cut (read opto back with the *?4* command). If the opto was not at the state expected, steps may have been lost.

Step loss detection can also be done by looking for changes on the other inputs.

Appendix 4. Device response packet

Introduction

EZSteppers® and EZServos® respond to commands by sending messages addressed to the "Master Device." The master device (which is typically a PC) is always assumed to have address zero (0). The master device should parse the communications on the bus continuously for responses starting with */0*. (It should NOT, for example, look for the next character coming back after issuing a command, because glitches on the bus when the bus reverses direction can sometimes be interpreted as characters.)

Response packet structure

After */0,* next comes the "Status Character" which consists of 8 bits:

Bit7 Reserved

Bit6 Always set

Bit5 Ready bit. Set when the EZStepper® or EZServo® is ready to accept a command.

Bit4 Reserved

Bits 3 through 0. These form an error code N from 0-15:

N	Function
0	No Error
1	Init Error
2	Bad Command (illegal command was sent)
3	Bad Operand (Out of range operand value)
4	N/A
5	Communications Error (Internal communications error)
6	N/A
7	Not Initialized (Controller was not initialized before attempting a move)
8	N/A
9	Overload Error (Physical system could not keep up with commanded position)
10	N/A
11	Move Not Allowed
12	N/A
13	N/A
14	N/A
15	Command overflow (unit was already executing a command when another command was received)

Note that in the RS485 bus, devices must respond right away, after the master sends a command, before the success or failure of the execution

of the command is known. For this reason, some error messages that come back are for the previous command. An example of this is “failure to find home.”

Example initialization error response

Note that the upper nibble typically only takes on values of 4 or 6 (Hex).

An initialization error response has 1 in the lower nibble. So the response is 41 Hex or 61 Hex which corresponds to ASCII character upper case “A” or lower case “a,” depending on whether or not the device is busy.

Example invalid command response

Note that the upper nibble typically only takes on values of 4 or 6 (Hex)

An invalid command response has 2 in the lower nibble. So the response is 42 Hex or 62 Hex, which corresponds to ASCII character upper case “B” or lower case “b,” depending on whether or not the device is busy.

Example operand out of range response

Note that the upper nibble typically only takes on values of 4 or 6 (Hex).

An operand out of range response has 3 in the lower nibble. So the response is 43 Hex or 63 Hex, which corresponds to ASCII character upper case “C” or lower case “c,” depending on whether or not the device is busy.

Example overload error response

Note that the upper nibble typically only takes on values of 4 or 6 (Hex).

An overload error response has 7 in the lower nibble. So the response is 47 Hex or 67 Hex, which corresponds to ASCII character upper case “I” or lower case “i,” depending on whether or not the device is busy.

Example response to command “/1?4”

FFh: RS485 line turn around character. It is transmitted at the beginning of a message.

2Fh: ASCII “/” Start character. The DT (Data Terminal) protocol uses the ‘/’ for this.

30h: ASCII “0” This is the address of the recipient for the message.

In this case ASCII zero (30h) represents the master controller.

60h: This is the status character (as explained above).

31h: These two bytes are the actual answer in ASCII.

This is an eleven which represents the status of the four inputs.

The inputs form a four-bit value. The weighting of the bits is:

Bit 0 = Switch 1

Bit 1 = Switch 2

Bit 2 = Opto 1

Bit 3 = Opto 2

03h: This is the ETX, or end-of-text character. It is located at the end of the answer string.

0Dh: This is the carriage return.

0Ah: This is the line feed.

Appendix 5. Microstepping primer

First consider a full stepping driver.

A stepper motor moves by having two windings that are orthogonal to each other and sequencing the current in these windings.

When full stepping, a typical sequence is:

- A+ (Only winding A current applied in positive direction)
- B+ (Only winding B current applied in positive direction)
- A- (Only winding A current applied in negative direction)
- B- (Only winding B current applied in negative direction)

(A full electrical cycle consists of four steps.)

It can be seen that if the windings are not physically placed orthogonally, the four steps may not be of equal size, and the difference in motion will be a constant only if the number of steps is divisible by four, even when in full step mode.

Now consider microstepping:

Microstepping is achieved by placing two sinusoidally varying currents that are 90 degrees apart in the windings of the stepper. This causes a torque vector of equal length to rotate, causing smooth inter-step motion of the rotor.

However, in order to get even motion in every step it is necessary:

- That the windings be mechanically orthogonal
- That the windings produce equal torques for equal currents
- That there is no other “detent torque” acting upon the rotor in the absence of current. (This detent torque is easily felt by rotating the stepper with windings disconnected and not shorted. A motor that is good for microstepping will feel smooth when rotated by hand—somewhat like a DC motor—with little tendency to detent.)
- That the current not be so small that the driver cannot regulate it to the microstepping accuracy desired

In general, most inexpensive stepper motors cannot microstep with accuracy. Typically, a motor designed especially for microstepping must be run at a significant current in order to get even microsteps. When accuracy is required, the move current must generally be set equal to the hold current. This is because if the current is reduced at the end of the move, the motor will fall back into a detent position.

Appendix 6. Stepper motor electrical specification

The EZStepper® will work with most stepper motors. However, the performance achieved will be a function of the motor used.

A stepper motor moves by generating a rotating magnetic field, which is followed by a rotor. This magnetic field is produced by placing a sine wave and a cosine wave on two coils that are spaced 90 degrees apart. The torque is proportional to the magnetic field, and thus to the current in the windings.

As the motor spins faster, the current in the windings needs to be changed faster in a sinusoidal fashion. However the inductance of the motor will begin to limit the ability to change the current. This is the main limitation on how fast a motor can spin.

Each winding of the motor can be modeled as an inductor in series with a resistor. If a step in voltage is applied, the current will rise with time constant L/R. If L is in Henrys and R is in ohms, then L/R is the time it takes in seconds for the current to reach 63% of its final value. (NOTE: there is also the back EMF of the motor, which essentially subtracts from the applied voltage.)

The current *I* for a step function of voltage *V* into a coil is given by:

I = (V/R) (1-^(tR/L))

This equation is a standard response of a first-order system to a step input. The final value of current is seen to be V/R. (This system is similar to a spring (L) in parallel with a damper (R) being acted upon by a step in force (V) giving a resulting velocity (I).)

Maximizing speed at which current can be changed

There are two methods by which the current can be made to change faster:

1. Reduce the inductance of the motor.

2. Increase the forcing function voltage V.

For (1) it is seen that for high performance, a motor with low inductance is desired.

For (2) the trick is to use a motor which is rated at about ¼ of the supply voltage (V). This minimizes the time it takes to ramp the current to a given value. (Once the current reaches the desired value, the “chopper” type drive used in the EZSteppers® will “chop” the input voltage in order to maintain the current—so the current never actually gets to the final value of V/R, but the advantage of “heading towards” a higher current with the same time constant is that the current gets to any given value faster.) In addition, using a lower voltage motor results in less back EMF, and does not subtract as much from the applied voltage.

So, for example, for a 24V supply, use a motor rated at around 6V, and then use the *m* and *h* commands to set the current regulation at or below

the rating for the motor. The default values on power-up are *h*=10% and *m*=25%, and should be safe for most motors.

EZ4AXIS operation

The EZ4AXIS17XR will drive at 0.25A per phase when *m*=50 (peak of sinusoidal drive move current waveform) with no restrictions.

However, at 0.5A per phase (peak of sine wave) when *m*=100, the drive can only be operated at about 25% move time with rests in between at a low hold current. Typically the move is limited because of thermal considerations to a maximum of approximately one minute continuous.

Maximizing power to motors on EZ4AXIS17XR

In order to exploit the full capability of the EZ4AXIS17XR, the motors must be of a low resistance (less than 5Ω or so) and the supply voltage—as mentioned above—should be about 4X the motor voltage. So use a 6V motor with a 24V supply. This is necessary not only for the reasons described above, but also because the maximum input current from the power supply is restricted to 2A. So input power is about 24V x 2A = 48W. The drive will step down the voltage and step up the current, much like a switching supply. So when using 6V motors, a total of 48W/6V= 8A is available to drive the eight phases of the four motors. Thus it is possible to get 0.5A amps per phase on four motors with the input current being only two amps at 24V .

See also Appendix 7, “Heat Dissipation (EZStepper® products with motor drives)” beginning on page 95.

AllMotion® provides a heat sink for this drive. Please see stepper accessories on the AllMotion® website:
http://www.allmotion.com/stepperaccessories.htm

Summary

Motor/supply voltage requirements are affected by inductance, resistance, and back emf. In order to obtain maximum current rise and available power, the following are recommended:

- Use motors with low inductance and resistance (5Ω or less).
- Use motors with low voltage rating (e.g., 6V).
- Use power supply at 4X motor voltage (e.g., with 6V motors use a 24V supply).
- Set move and hold current values using *h* and *m* commands if change from default is needed.

Appendix 7. Heat dissipation (EZStepper® products with motor drives)

Overview

Most stepper applications require intermittent moving of the motor. In the EZStepper®, the current is increased to the move current, the move is performed, and the current is then reduced to the hold current (automatically). The dissipation in the drive is proportional to the current flowing in the drive, and therefore the dissipation occurs primarily during the move.

When the drive generates heat, the heat first warms the circuit board and heat fin (ordered separately if needed)

Only then does the heat transfer to the surroundings. For intermittent moves that are less than one minute in duration, the drive primarily cools using this thermal inertia of the board and heat fin, and not by steady state dissipation to the surrounding ambient.

Running at high current/duty cycle

The electronics for EZSteppers® are fully capable of running at the rated voltage and current. However, due to the small size of the boards, which limits the steady state heat transfer to the ambient, care must be taken when the drive is used in high duty cycle and/or high current applications. For conservative operation, it is recommended that the duty cycle be reduced linearly, from 100% duty at 50% of rated current, to 25% duty at 100% of rated current. (Duty cycle means the percentage of the time that the drive is moving the load, averaged over 5 minutes). Conservatively, the maximum continuous run at 100% current is about one minute. An on-board thermal cutout typically trips after about two minutes at 100% current. (This cutout is self-resetting when the drive cools). Of course, at 50% of current, the drive will run continuously with no time limit.

Most “intermittent move” applications will NOT require derating of the drive.

Typically if using the motor at high current and high duty cycle (move time), please purchase the additional heat sink.

In addition, if running high current on two motors, select non-adjacent channels (1 and 3, for example) to distribute the heat load within the board.

EZSteppers® are designed with parts rated at 85° C or better. This means the PCB copper temperature must remain below 85° C. The ambient air temperature allowed depends on the airflow conditions.

MTBF is 20,000 hr. at 85° C PCB copper temperature, and doubles for every 10° C under 85° C.

Appendix 8. OEM Protocol with checksum

Introduction

The protocol described in the majority of this manual is DT (Data Terminal). There is, however, a more robust protocol known as OEM that includes checksums. AllMotion, Inc. drives work transparently under both protocols, and switch between the protocols depending on the start transmission character detected.

The OEM protocol uses 02 hex (Ctrl B) as the start character, and 03 Hex (Ctrl C) as the stop character. The 02 Hex start character is equivalent to the / character in DT protocol.

OEM Protocol example 1

/1A12345R in DT protocol is equivalent to

(CtrlB)11A12345R(Ctrl C)# in OEM protocol.

Name	**Typed**	**Hex**
Start Character	*Ctrl B*	02
Address	*1*	31
Sequence	*1*	31
Command	*A*	41
Operand	*1*	31
Operand	*2*	32
Operand	*3*	33
Operand	*4*	34
Operand	*5*	35
Run	*R*	52
End Character	*Ctrl C*	03
Checksum	#	23

The checksum is the binary 8-bit XOR of every character typed, including the start and end characters. (The sequence character should be kept at 1 when experimenting for the first time.) Note that there is no need to issue a carriage return in OEM protocol.

OEM Protocol example 2

/1gA1000M500A0M500G10R in DT protocol is equivalent to *(CtrlB)11gA1000M500A0M500G10R(CtrlC)C* in OEM protocol.

The C at the end is Hex 43, which is the checksum (binary XOR of all preceding bytes).

Sequence Character:

The Sequence Character comes into effect if a response to a command is not received from the drive. In this instance the same command can be resent with bit 3 (repeat bit) of the sequence byte set, and bits 0-2 representing the sequence number.

When the repeat bit is set consecutive commands received by the drive must have a different sequence number in order to get executed. Only the sequence number is looked at—not the command itself—in determining whether the command should be executed. So, if the drive has already seen this command sequence (the value in bits 0-2), it will not execute it again, but will acknowledge (again) that the command was received.

This covers both possibilities that (a) the drive didn't receive the command, and (b) the drive received the command but the response was not received.

The sequence number can take the following values:

- 31-37 without the repeat bit set
- 39-3F with the repeat bit set

(The upper nibble of the sequence byte is always 3.)

Appendix 9. Linear and circular interpolation

Drawing circles and lines – NOT IMPLEMENTED YET IN EZ4AXIS17XR – ONLY IMPLEMENTED IN EZ4AXIS

The EZ4AXIS is capable of coordinated motion among axes. Interpolation commands directly coordinate the motion of multiple axes for specific tasks, and are available for drawing circles and straight lines.

Overview

Interpolation commands control Axes 1 and 2 simultaneously. To implement interpolation commands, Axes 1 and 2 drive an X-Y type mechanism such as shown here:

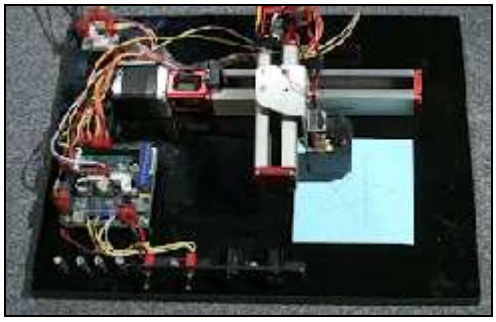


Figure 7 X-Y Mechanism for Interpolation Commands

You can see this mechanism in action in the demo video located at http://www.allmotion.com/Flash_Video_Pages/Example_Videos/demos_examples_4AXIS_Linear_flash.html

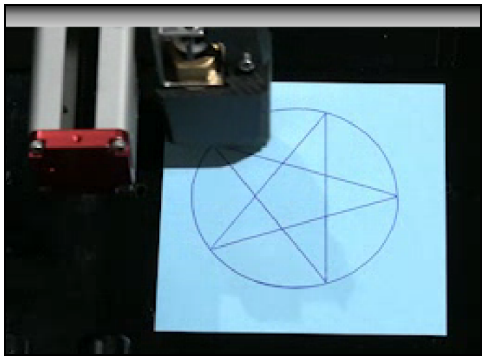


Figure 8 Circular and Linear Interpolation

Circular interpolation

There are four commands associated with circular interpolation:

aaA Sets the diameter of the circle in microsteps (0 – 90000 microsteps)

aaI Sets beginning phase of the circle (0-1024) (degrees/360*1024)

aaW Sets the speed at which the circle is drawn in microsteps/second (1-20000). Note: speed needs to be lower for larger diameter circles.

aaC Sets arc (how much of a circle is drawn) and initiates the circle drawing process (1-1024) (degrees/360*1024). Note: *aaC2048* will draw two circles on top of one another.

The following figure illustrates how the commands define the circle.

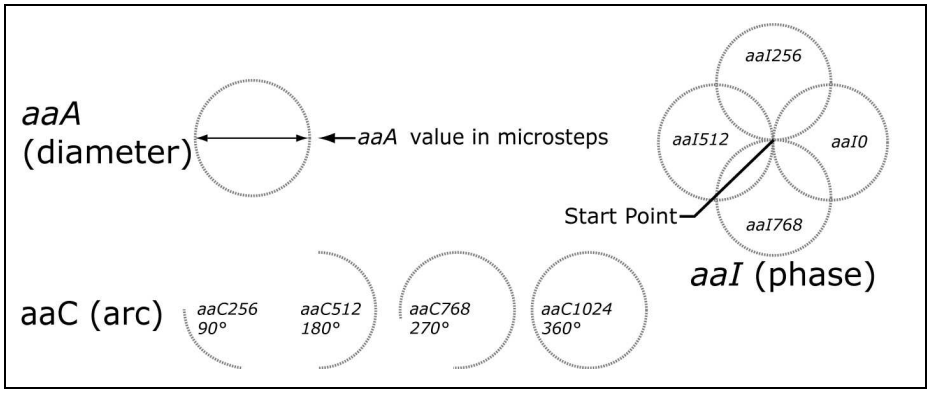


Figure 9 Circular Interpolation Command Effects

To draw a circle, issue a command such as:

/1aM1A40070,35200aaW1000aaI0aaA14000aaC256R

This results in an arc, or part of a circle, that is drawn at a speed of 1000 microsteps/second (*aaW1000*), begins at 0° (*aaI0*), has a diameter of 14000 microsteps (*aaA14000*), and spans an arc of 90° (*aaC256*).

NOTES

- Place commands in the order shown above.
- Do not add commands for axes 3 and 4 when issuing circular interpolation command strings.
- Do not include circular interpolation commands with other commands in the same string.
- Issue the *aaC* command last, since it initiates the circle drawing process.

Linear interpolation

Overview

There are two commands associated with linear interpolation:

an65536 TBD Edit Enable linear interpolation. The *an* command uses weighted bit values just as the *n* mode commands do.

an0 Exit linear interpolation mode.

Once linear interpolation has been enabled, use the multi-axis *A* command (move to absolute position), for example:

A1000,2000R (e.g., */1an65536 A1000,2000R*)

This will cause Axis 1 to move to position 1000, and Axis 2 to move to position 2000. The two moves are tied together so that the pen driven by the two axes moves in a straight line. This is also demonstrated in the video referred to above.

To move to multiple interpolated positions, use the A command for each interpolated position. For example, notice *A3500,6000* in this command string:

/1an65536 A1000,2000,A3500,6000R

NOTE: You may issue a multi-axis command addressed to all four axes while the linear interpolation mode is on. In such a case Axes 1 and 2 will move in interpolation fashion, while Axes 3 and 4 will move normally.

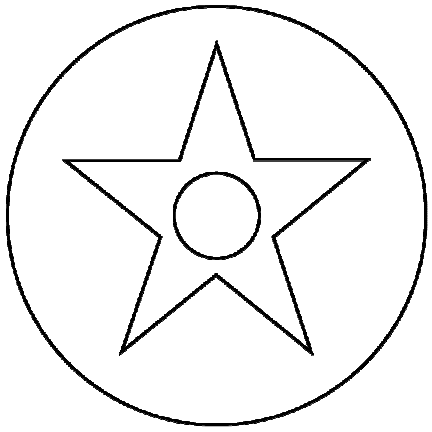
Exiting linear interpolation mode

To exit linear interpolation mode, enter the command *an0* following the move commands, e.g., */1an65536 A1000,2000,A3500,6000an0R* or */1an0R*.

Circle and star example

Introduction

This example describes how an EZ4AXIS was programmed to draw a composite star-circle pattern utilizing a dual-axis stepper motor mechanism equipped with ink pen actuated by a solenoid. Here is the example pattern:



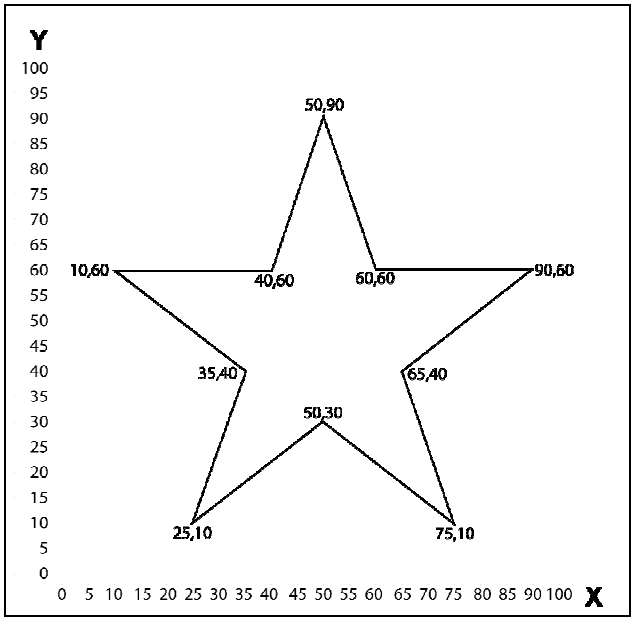
This pattern was implemented by five command strings, each stored in a different memory location in the on-board EEPROM:

- Startup instructions, which set basic operating parameters and begins the sequence when a high appears at Switches 1 and 2. (Memory location 0, which runs automatically at power-up.)
- Draw star pattern utilizing linear interpolation. (Memory location 4)
- Return both axes home. (Memory location 7)
- Draw small circle pattern utilizing circular interpolation, and return both axes home. (Memory location 5)
- Draw large circle pattern utilizing circular interpolation, return both axes home, and then execute the string in memory location zero (startup instructions). (Memory location 6)

  At this point the equipment is ready to draw the pattern again when a high appears at Switches 1 and 2.

The star pattern

The following illustration shows the X-Y coordinates for points on the star in a grid with a scale of 0-100. The X coordinate is written first, then Y. Zero (0) represents the home positions of the two axes.



The coordinates are the starting and stopping points for the pen.

The X-Y numbers are translated proportionally into microsteps according to the desired size of the star. This star was intended to fit into a space of 64000 microsteps, comfortably below the 70000 microstep range of the fixture in use.

So each of the coordinates in a pair becomes a percentage of the 64000 microstep range (coordinate/100*64000 = coordinate in microsteps).

A 3200 microstep offset was then added to each of the coordinates to place the star pattern well out of the way of the home positions of the two axes.

For example, the coordinate 60 would be 60/100*64000+3200=41600.

Command string for star pattern (location 4)

The coordinate pairs in the diagram above are shown in **bold**.

*/1s4aM1V12000aM2V12000**A60800,35200**an65536J3**A41600,28800,A41600,9600,A28800,25600,A9600,19200,A22400,35200,A9600,51200,A28800,44800,A41600,60800,A41600,41600,A60800,35200**J0an0e7R*

Breakdown:

/1 Select Drive 1.

s4 Store the following commands in memory location 4.

aM1V12000aM2V12000 Set velocities of Axes 1 and 2 to 12000 microsteps/second.

A60800,35200 Move Axes 1 and 2 to absolute positions comprising the starting point for drawing the star.

an65536 Enter the linear interpolation mode.

J3 Turn ON/OFF drivers on, engaging the pen solenoid.

A41600 . . . 35200 Execute moves sequentially to absolute position defined by each coordinate pair (e.g., *A41600,28800*) using the absolute move command (*A*).

J0 Turn ON/OFF drivers off, releasing the pen solenoid.

an0 Exit the linear interpolation mode.

e7 Execute string 7 (the program in memory location 7), the homing command string. Note that this is stored in a separate location from the star pattern to avoid exceeding the capacity of the memory location.

R Run the command string.

Command string for homing after drawing star pattern (location 7)

/1s7aM1V30000f1Z200000aM2V30000f1Z200000e5R

Breakdown:

/1 Select Drive 1.

s7 Store the following commands in memory location 7.

aM1V30000f1Z200000 On Axis 1, set velocity (*V*) to 30000 microsteps/second; set polarity of home flag to normally open (*f1*); and go home (*Z*), allowing 200000 microsteps to reach home.

aM2V30000f1Z200000 Same as above for Axis 2.

e5 Execute the contents of memory location 5 (the small circle pattern).

R Run the command string.

Circle patterns

- The circles are both aligned with the center of the star on the Y axis.
- The radius is the distance, in microsteps, from the center of the star to the desired circumference. Multiplied by two, this is the diameter used in the command string.
- The remaining commands are described on a previous page.
- The circle command string should have commands placed in the order shown and include a command to home both axes. A velocity command is included, noting that the velocity must be lower for successfully drawing larger circles.

Command string for smaller circle (location 5)

/1s5aM1V8000aM2V8000aM1A40070,35200J3aaW1000aaI256 aaA14000aaC1024J0aM1V30000f1Z200000aM2V30000f1Z200000e6R

Breakdown:

/1 Select Drive 1.

s5 Store the following commands in memory location 5.

aM1V8000aM2V8000 Set velocities of Axes 1 and 2 to 8000 microsteps/second.

aM1 Select Axis 1.

A40070,35200 Move Axes 1 and 2 to absolute positions comprising the starting point for drawing the circle.

J3 Turn ON/OFF drivers on, engaging the pen solenoid.

aaW1000 Set speed of drawing to 1000 microsteps/second.

aaI256 Set beginning phase of circle to 90 degrees.

aaA14000 Set diameter of circle to 14000 microsteps.

aaC1024 Draw a full 360 degree circle. This command also starts the circular interpolation mode.

J0 Turn ON/OFF drivers off, releasing the pen solenoid.

aM1V30000f1Z200000 On Axis 1, set velocity (*V*) to 30000 microsteps/second; set polarity of home flag to normally open (*f1*); and go home (*Z*), allowing 200000 microsteps to reach home.

aM2V30000f1Z200000 Same as above for Axis 2.

e6 Execute the contents of memory location 6 (larger circle pattern).

R Run the command string.

Command string for larger circle (location 6)

/1s6aM1V12000aM2V12000aM1A33070,67200J3aaW1000aaI0 aaA64000aaC1024J0aM1V30000f1Z200000aM2V30000f1Z200000e0R

Breakdown:

/1 Select Drive 1.

s6 Store the following commands in memory location 6.

aM1V12000aM2V12000 Set Axes 1 and 2 velocities to 12000 microsteps/second.

aM1 Select Axis 1.

A33070,67200 Move Axes 1 and 2 to absolute positions comprising the starting point for drawing the circle.

J3 Turn ON/OFF drivers on, engaging the pen solenoid.

aaW1000 Set speed of drawing to 1000 microsteps/second.

aaI0 Set beginning phase of circle to 0 degrees.

aaA64000 Set diameter of circle to 64000 microsteps.

aaC1024 Draw a full 360-degree circle. This command also starts the circular interpolation mode.

J0 Turn ON/OFF drivers off, releasing the pen solenoid.

aM1V30000f1Z200000 On Axis 1, set velocity (*V*) to 30000 microsteps/second; set polarity of home flag to normally open (*f1*); and go home (*Z*), allowing 200000 microsteps to reach home.

aM2V30000f1Z200000 Same as above for Axis 2.

e0 Execute command string stored in memory location 0 (startup command string).

R Run the command string.

Command string for startup (location 0)

This is the command string designed to execute at power-up, since it is stored in memory location 0.

/1s0aM1m30L10V5000f1Z200000aM2m30L10V5000f1Z200000H01H02 M100e4R

Breakdown:

/1	Select Drive 1.
s0	Store the following in memory location 0.
aM1	Select Axis 1.
m30	Set move current to 30% of max (2A).
L10	Set acceleration factor to 10.
V5000	Set velocity (*V*) to 5000 microsteps/second.
f1	Set polarity of home flag to normally open (*f1*).
Z200000	Go home (*Z*), allowing 200,000 microsteps to reach home.
aM2m30L10V5000f1Z200000	Same as above for Axis 2.
H01	Halt and wait for 0 on Switch 1.
H02	Halt and wait for 0 on Switch 2.
M100	Wait 100 milliseconds.
e4	Execute the contents of memory location 4 (the star pattern).
R	Run the command string.