Commander core
CMD-4CR

Advanced 4-Axis
Motion Controller Core

First edition, October 2019

NIPPON PULSE AMERICA, INC. makes no representations or warranties regarding the content of this document. We reserve the right to revise this document any time without notice and obligation.

Revision History

Date and Revision	Firmware Compatibility	Changes Made
October 2019 Version 1.0	V126BL	New Document

Cautions

Copying all or any part of this manual without written approval is prohibited.

The specifications of this controller may be changed to improve performance or quality without prior notice.

Although this manual was produced with the utmost care, if you find any points that are unclear, wrong, or have inadequate descriptions, please let us know.

We are not responsible for any results that occur from using this controller, regardless of item (3) above.

The Commander core is designed for use in commercial apparatus (office machines, communication equipment, measuring equipment, and household appliances). If you use it in any device that may require high quality and reliability, or where faults or malfunctions may directly affect human survival or injure humans, such as in nuclear power control devices, aviation devices or spacecraft, traffic signals, fire control, or various types of safety devices, we will not be liable for any problem that occurs, even if it was directly caused by the Commander core. Customers must provide their own safety measures to ensure appropriate performance in all circumstances.

Explanation of the descriptions in this manual

The "X" "Y" "Z" and "U" of terminal names and bit names refer to the X-axis, Y-axis, Z-axis, and U-axis, respectively.

Terminals with a / (ex. /RST) are negative logic. Their logic cannot be changed. Terminals without a / are positive logic. Their output logic can be changed.

When describing the bits in registers, "n" refers to the bit position. A "0" means that the bit is in position 0 and that it is prohibited to write to any bit other than "0." Finally, this bit will always return a "0" when readout.

1.0 Introduction and Overview

Commander motion controller core (CMD-4CR) is a first in class advanced 4 axes, standalone-programmable motion controller hybrid IC. The Commander core design provides OEMs the convenience of an off-the-shelf controller and the technology of a designed-from-scratch controller. As such it exhibits the best qualities of both types of controllers:

- Faster path-to-market
- Flexible design that fits a large range of applications
- No sacrifice to controller capabilities and processing speed
- Easily scalable from prototype to production with no software changes required
- Simple to use
- Cost-effective

The Commander core architecture allows for easy integration into custom hardware. Communication to the Commander core can be established over USB, Serial (RS-485, I^{2}C, SPI) or Ethernet. It is possible to download four standalone programs to the device and have it run independently of a host.

The Commander core is a CMOS hybrid IC designed to provide the oscillating, high-speed pulses needed to drive stepper motors and digital servomotors (pulse string input types) using various commands. It can offer various types of control over the pulse strings and therefore the motor performance. These include continuous feeding, positioning, and origin return, at a constant speed, and linear or S-curve acceleration/deceleration.

The Commander core controls four axes. It can control the linear interpolation of two to four axes, circular interpolation between any two axes, confirm controller operation status, and output an interrupt with various conditions. It also integrates an interface for servo motor drivers.

These functions can be used with simple commands. The intelligent design philosophy reduces the burden on the CPU. Commander is a robust and powerful controller with numerous operating capabilities. This table summarizes a variety of capabilities and features available in the Commander core software:

Feature	Description
Standalone control separated from PC	Operate motion without PC using standalone functionality
Programming language for stand-alone program with utility software	A-Script programming language, develop program on PC, compile and download to Commander
Operation check by utility software	Easy to use monitoring screens for motor status, I/O status, and system operation status
Compile, write and read standalone program by utility software	Easy to use tools to write, edit and debug software
Joystick operation with analog input (X-axis, Y-axis)	Utilize joystick for manual control of motion
13 types of homing mode	Flexibility in how machine mechanisms are set-up and homing operation is initiated
Manual pulse generator operation	
On-the-fly speed change (speed overridden during movement)	Quickly change operation speed manually on-the-fly
On-the-fly target position change (target position overridden during movement).	Target position can be updated manually on-the-fly
2 to 4-axis linear interpolation	Create a variety of paths utilizing interpolation options
2-axis circular and arc interpolation, including 3rd axis for Helical motion	Create circles, arcs or even create helical motion utilizing x, y, z axis
Sync output configuration	
Absolute positioning or incremental positioning can be selected	Flexibility of positioning coordinates as absolute from a home location or incremental
12 inputs and 12 outputs of general-purpose input/output signals	Flexibility in outputs, including configuration of general-purpose I/O, dedicated I/O and high-speed inputs

1.1 Product Overview

The Commander core is based on the PCL6045BL motion control LSI, and an ARM Core processor.

Power

- +3.3VDC Power Input
- 3.3V logic level, 5V tolerant

Communication

- USB 2.0, HID compatible. No driver signature required
- RS-485 ASCII, selectable 9600, 19200, 38400, 57600, 115200 baud rates available
- I^{2}C bus [1 channel], available for future expansion for interface with external IC
- SPI bus [2 channels], available for future expansion for interface with external IC
- Ethernet, available for future expansion

Control

- Command control from PC through USB or serial communication
- Standalone control allows operation separated from PC, using a BASIC-like programming language known as A-SCRIPT

Utility software

- Allows for operation check by utility software
- Compile, write and read standalone programs

Motor interface

- Maximum pulse output rate of 6.55M PPS
- Stepper motor interface with pulse, direction, and enable outputs for XYZU
- Servo motor interface with pulse, direction, enable, in-position, servo alarm, and error clear outputs for XYZU

Homing Routines

- 13 different built-in homing routine available

Positioning operations

- Selectable trapezoidal or s-curve acceleration
- Absolute positioning or incremental positioning can be selected
- On-the-fly speed change (speed overridden during movement).
- On-the-fly target position change (target position overridden during movement).
- Coordinated motion
    - Linear (XYZU) (2 to 4 axis)
    - Circular/Arc (any two axes)
    - Helix/Tangential (XYZ)
    - Continuous linear/circular/arc coordinated buffered move for XYZ axis
- Control using manual pulse generator (MPG)
    - Electronic camming
    - Electronic gearing

Inputs/Outputs

- With integrated noise filter, to reduce switch bounce
- A/B/Z encoder inputs for XYZU (TTL Compatible)
    - Maximum frequency of 6.5 MHz (26M counts with 4x Multiplication)
    - StepNLoop closed-loop position verification algorithm (XYZU)
- Manual pulse generator (MPG) operation
    - A/B pulse inputs for XYZU (TTL Compatible)
    - Maximum frequency of 6.5 MHz (26M counts with 4x Multiplication)
- Digital I/O
    - 4 designated highspeed inputs
    - 4 designated highspeed outputs
    - +Limit/-Limit/Home inputs for XYZU
    - Simultaneous start input
    - Emergency stop input
    - 32 configurable I/O
    - Latching input
    - Synchronization Pulse output
- Analog I/O
    - Two 10-bit analog input
    - Two PWM outputs
    - Joystick operation with analog input (X-axis, Y-axis)

Firmware upgrade and download via Boot-loader through USB

1.2 Specifications

Item	Description
Number of Axes	4 axes (X, Y, Z, and U axis)
Positioning control range	-134,217,728 to +134,217,727 (28-bit)
Ramping-down point setting range	0 to 16,777,215 (24-bit)
Number of registers used for setting speeds	Three for each axis (FL, FH, and FA (speed correction))
Speed setting step range	1 to 65,535 (16-bits)
Speed magnification range[1]	Multiply by 0.1 to 100 Multiply by 0.1 = 0.1 to 6,553.5 pps Multiply by 1 = 1 to 65,535 pps Multiply by 100 = 100 to 6,553,500 pps
Acceleration/deceleration characteristics	Selectable as Linear or S-curve acceleration/deceleration.
Acceleration rate setting range	1 to 65,535 (16-bit)
Deceleration rate setting range	1 to 65,535 (16-bit)
Feed speed automatic correction function	Automatically lowers the feed speed for short distance positioning moves.
Manual operation input	Manual pulse generator (MPG) input, pushbutton switch input
Interpolation functions	Linear interpolation: Any 2 to 4 axes, Circular interpolation: Any 2 axes
Operating temperature range	-40 to +85°C
Power	Single power supply of 3.3 V±10%
Package	150-pin Hybrid IC

[1] See **Appendix B: Speed Settings**

2.0 Operating principle

The Commander core has many modes of operation. These include single, multi-axis, interpolation, on-the-fly speed or target position changes, status monitoring functions, and I/O operations. Any or all of the different modes of operation can occur simultaneously without affecting the performance of the other operations.

Single-axis operations involve motion of a single axis and does not require interaction with any other axes. **Multi-axis** operations involve motion of two or more axes but does not require interaction between the axes. **Interpolated** operations involve two or more axes whose movements are interdependent. There are several types of interpolated moves. **Linear interpolation** is when two or more axes work together to move along the hypotenuse between the starting and target coordinates. For linear interpolation, each axis moves in only one direction and start and stop at the same time. **Circular interpolation** includes two-axes working together to move along the outer rim of a circle. For circular interpolation, each axis will change direction and speed as required to allow for a circular path of motion. A popular subset of circular interpolation is **arc interpolation** where two axes work together to move along the outer rim of a circle for a set number of degrees. **Helical or tangential interpolation** is when circular or arc interpolation of the X and Y-axis is combined with linear interpolation of the center point of the circle and the Z-axis. This allows for movement along a helical motion path.

The Commander core allows for all of these different moves:

- as independent moves
- any combination of the above moves
- or **buffered** together to allow for seamless **continuous interpolated** motion.

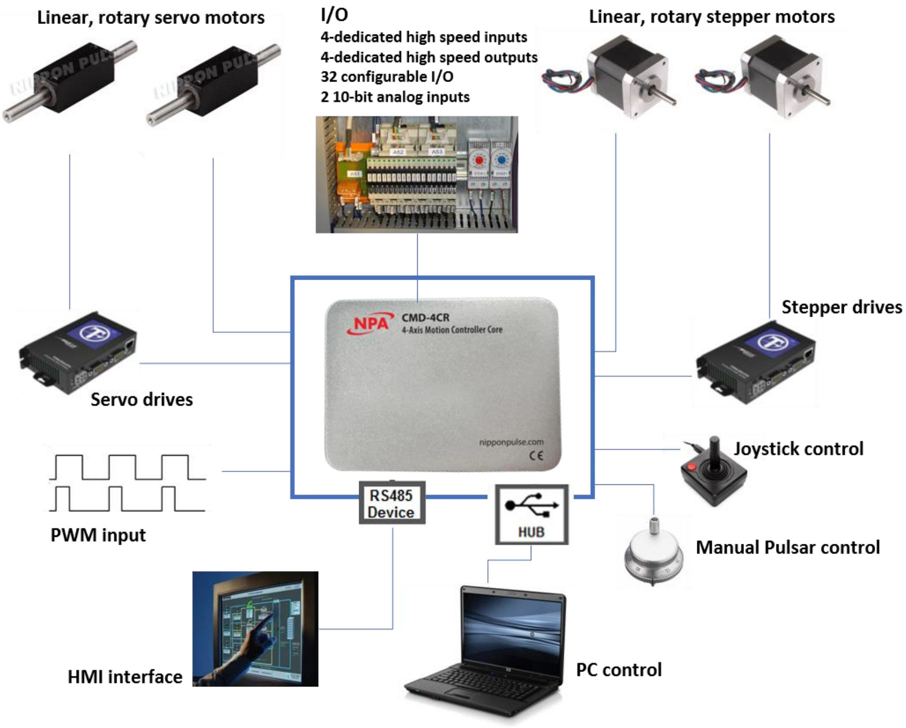
When operating a motor, it is possible to **jog** a motor at speed until it is told to **stop** or move to a predefined target **position**. The target position can be **absolute** to a 0 (home) point defined during one of the many **homing** processes or can be **incremental** or relative to the motor's current position. While moving it may be necessary to change the **target speed,** or while making a positioning move, it may be necessary to change **target position.** The Commander core can seamlessly do this without requiring any of the motors to stop. These operations are referred to as **on-the-fly speed change** or **on-the-fly target position change** respectively**.** It is also possible to use a **joystick** or **analog input** to control a motor. Using the **MPG** function, the motor can follow a master axis or a cam.

Regardless of the type of operation, the first step is to define your motion profile. The basics of a motion profile are: acceleration/deceleration type (**S-curve** or **Linear**/trapezoidal), the initial or **low speed,** the max or **high speed**, and the amount of time needed for **acceleration** and **deceleration**.

There are a number of inputs and outputs available on the Commander core. These can be broken up into two categories; dedicated high-speed (ηS) and general-purpose. Dedicated inputs/outputs do not require further interaction from the microprocessor, and usually fall into one of two categories: 1) Inputs/Outputs that automatically cause a reaction that is fixed, and 2) Inputs/Outputs that automatically cause a reaction that can be configured. Safety inputs such as **end limits**, **slowdown** (near the limit) inputs, **home** (origin) switch, **latching** inputs, and **synchronization** outputs are all dedicated inputs. On the other hand, the **general-purpose** outputs will change state and require the microprocessor to detect and then act on their status. In addition, the microprocessor can request the current status of any input and of the various counters, output pulse, encoder, MPG, etc. along with the full status of each axis.

2.1 General Connections and Setup

Important Note: All the commands described in this section are ASCII commands unless called out as standalone commands. ASCII commands are used when communicating over USB or serial. Standalone commands are used when writing a standalone program to the Commander core. Not every ASCII command is supported in standalone operations.



2.1.1 Connecting multiple Commander cores in a system

2.1.1.1 Identification Number

The Commander core allows for each device to have a unique identification number, allowing up to 64 devices to be connected to the same host. If multiple devices are connected to the host, the identification number for each device should be unique.

By default, all Commander cores are shipped from the factory with identification number 01. The current identification number of a device can be found by reading the **ID** command. Also, the device name will always include the current identification number. For USB communication the device name will be "CMD-4CR-xx" where xx is the identification number of the device. For serial communication, the “DeviceName” is only the identification number of the device.

In order to change the identification number of the device, first store the desired number using the **ID** command. Note that this value must be within the range [00-99].

For example, the command **ID=02** can be sent to change the identification number to 02. To save a modified identification number to the flash memory of the device, use the **STORE** command. The new identification number will not take effect until after a power cycle.

The current version of firmware can be found with the **VER** command.

ASCII	ID	STORE	VER
Standalone	—	STORE	—

2.1.1.2 Synchronization with external trigger

It is possible to use an external trigger to start a defined motion profile. This is done by connecting the /CSTA pins as shown in Figure 2-1. The /CSTA line needs an external pullup resistor to VDD.

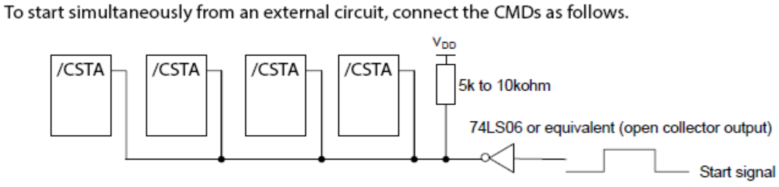

To start simultaneously from an external circuit, connect the CMDs as follows.

Figure 2-1

By default, the external start input is disabled (**EXST=0),** which will allow motion to start immediately upon receipt of any motion command. If the external start input is enabled (**EXST=1)** than any motion command sent will wait for the **/CSTA** input to go low, before motion is started. The Commander core can buffer up to 3 motions on a first-in first-out order; they will start one at a time each time the **/CSTA** input goes low.

The logical input of the **/CSTA** input cannot be changed.

ASCII	EXST
Standalone	—

2.1.2 Motor interface

The Commander core is designed to be a master controller for both stepper and digital servomotors and their drivers. There are a few differences between how a stepper and a servo system react and the connections needed between the Commander core and their respective drivers.

The difference between a stepper motor and a servomotor configuration is shown below. The design and construction of the motors are also different. Stepper motor operation is synchronized by command pulse signals output from a pulse generator (strictly speaking it follows the pulses). In contrast, a servomotor will translate the command pulse signals into speed, direction, and distance commands (usually one pulse equals one encoder pulse) as such the servomotor operation lags behind the command pulses.

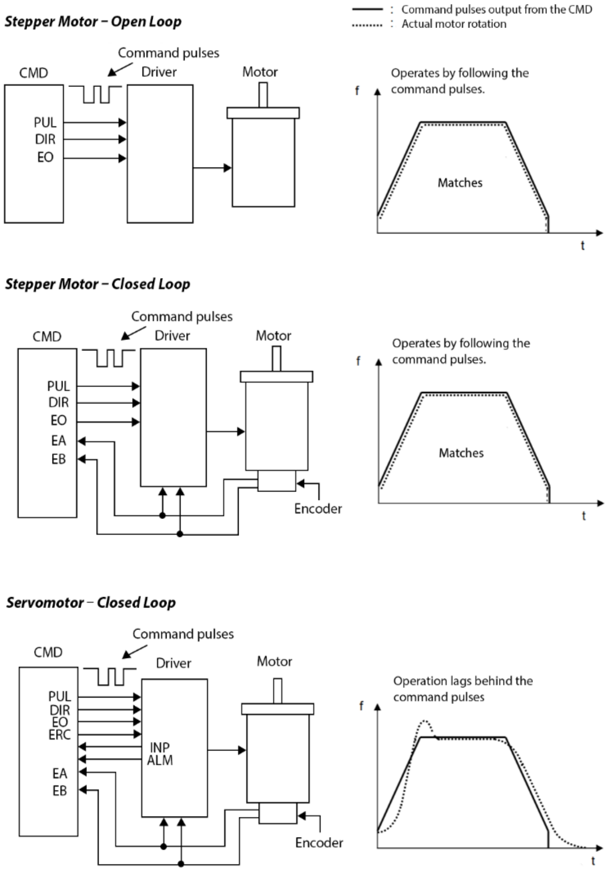

Stepper Motor – Open Loop
—— : Command pulses output from the CMD
········ : Actual motor rotation
CMD
Command pulses
Driver
Motor
PUL
DIR
EO
f
Operates by following the command pulses.
Matches
t
Stepper Motor – Closed Loop
CMD
Command pulses
Driver
Motor
PUL
DIR
EO
EA
EB
Encoder
f
Operates by following the command pulses.
Matches
t
Servomotor – Closed Loop
CMD
Command pulses
Driver
Motor
PUL
DIR
EO
ERC
EA
EB
INP
ALM
Encoder
f
Operation lags behind the command pulses
t

Figure 2-2

2.1.2.1 Stepper

Since a stepper motor operation is synchronized by command pulse signals output from the Commander core (strictly speaking it follows the pulses), the stepper driver will need the **pulse** (speed command), **direction** (direction command), and **enable** connected. An encoder connection is not required unless you will be using the **StepNLoop function**, however, there are advantages available if an encoder is connected.

The **pulse (PUL)** and **direction (DIR)** communicate the speed and direction in real-time to the drive. The polarity and output mode of these signals is set with the **POL[axis](Bit0~2)**.

The **enable (EO)** output is most often used to turn the motor excitation circuit ON/OFF. But if not required by your driver or application, it can be used as a general-purpose output. The polarity of this output is set with the **POL[axis](Bit9)**.

Figure 2-2 shows connection examples for both open loop and closed loop stepper systems.

ASCII	POL[axis]	EO
Standalone	—	EO

2.1.2.2 Digital Servo

The servomotor rotation lags behind the command pulses from the Commander core. This means that when the Commander core completes outputting pulses, the encoder will take some time to return all of the pulses. As such there is a need for more communication between the Commander core and the servo drive. A servomotor driver will need the **pulse** (speed command), **direction** (direction command), **enable**, **INP** (in-position), **ERC** (deflection counter clear) and **ALM** (alarm) connected. It is also strongly recommended to connect the encoder output to the Commander core.

The servomotor driver will send:

An **INP** (in-position) signal is sent from the driver when the motor has reached the requested position based on the pulses sent. The polarity of this input is set with the **POL[axis](Bit7)**. When the **INP[axis]** function is enabled the operation completion status is delayed until the in-position signal is received from the driver.

An **ALM** (alarm) signal is received when there is an error or abnormality in the servo driver or motor. This can include:

(1) The deflection amount becomes abnormally large.
(2) Excess current is flowing through the motor (instantaneous measurement, or at certain intervals)
(3) There is a temperature error.
(4) There is a power supply voltage error.

The alarm signal associated with each error varies with the servo driver. Generally, the signal from a servo driver is kept ON until it is cleared. The polarity of this input is set with the **POL[axis](Bit5)**.

As such the **ERC** command can be setup so the Commander core can send an **ERC** (deflection counter clear) signal to clear the error. The polarity of this output is set with the **POL[axis](Bit8)**, while the pulse width is set with the **ERCP** command and delay is set with the **ERCD** command.

Figure 2-2 shows connection examples for a servomotor driver.

ASCII	POL[axis]	EO	INP[axis]	ERC[axis]	ERCD[axis]	ERCP[axis]
Standalone	—	EO	—	—	—	—

2.1.2.3 Encoder Inputs

An encoder is a type of pulse generator that can be used to communicate to the Commander core the actual speed, direction, and position of the motor or system it is attached to. The Commander core accepts two different pulse types which are set using the **POL[axis](Bit11~12)**.

1) Two pulse type (not common) is where pulses are sent to A input to count up and B input to count down. Each pulse input is one count of the encoder.
    a. This can be used as an up/down counter to track pulses unrelated to the motion of the motor.

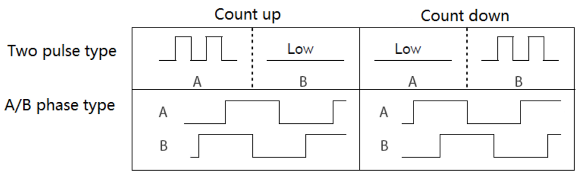


Figure 2-3

2) A/B phase type (common).
   a. For an A/B phase type there should be 90° difference between the pulses from A and B. See Figure 2-4.
   b. By default, when the A phase leads the B phase motion is in the positive direction, when B phase leads the A phase motion is in the negative direction. The direction can be changed by **POL[axis](Bit10)**. See Figure 2-5.
   c. For an A/B phase type you can select x1, x2 or x4 multiplication rate. Figure 2-6 shows how the encoder counts would be interpolated for a x1, x2, and x4 multiplication rate on a single encoder.
      i. x1 multiplication – Only the rising edge of the A phase is counted, B phase is not counted and only used to check the direction of motion.
      ii. x2 multiplication – The rising and falling edge of the A phase is counted, B phase is not counted and only used to check the direction of motion.
      iii. x4 multiplication – The rising and falling edge of the A and B phase is counted, and used to check the direction of motion.

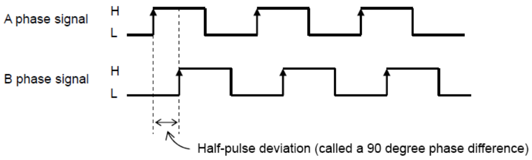


Figure 2-4

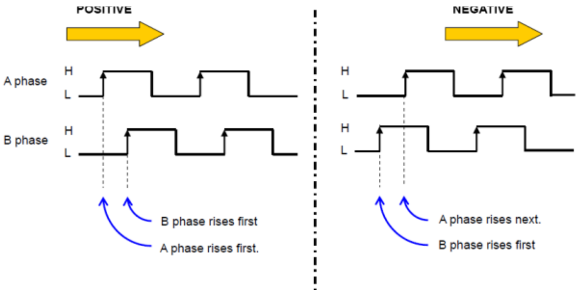


Figure 2-5

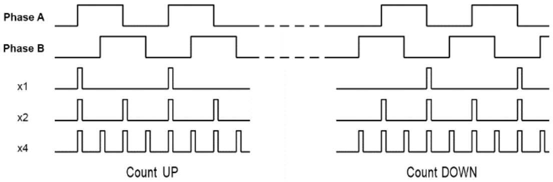


Figure 2-6

Maximum frequency input on the encoder line is the maximum pulses that can be input to an encoder input, not the counts. Since most encoders outputs are rated based on a x4 multiplication rate, the output frequency of pulses is most likely ¼ the published count rate output of the encoder. Please check the datasheet for your encoder to confirm.

While an encoder input is not required, there are a number of advantages to connecting one.

1) Closed loop control using StepNLoop – StepNLoop is a closed-loop position verification algorithm. When the encoder is placed at the work point, it can be used to eliminate backlash in the system. When the encoder is mounted to the motor, it can help overcome missed steps. Section 2.5.3 StepNLoop Closed Loop Control operation contains more information regarding StepNLoop.
2) Motor Stall detect – When an encoder is connected it is easy to detect if the motor has stalled.
3) Position verification – Verify the motor has finished it move and reached the desired position.
4) Z-index – the Z-index can improve your homing method. The polarity of Z-index input is set with the **POL[axis](Bit13)**

If an encoder is not connected or needed, the encoder inputs can be used as an up/down counter to track pulses unrelated to the motion of the motor.

ASCII	POL[axis]
Standalone	—

2.1.3 External Connections

The Commander core includes several types of inputs/and outputs which are not directly related to the motion of the motor. Digital I/O is broken down into three categories: 1) **Dedicated I/O** - high speed inputs or outputs which have a dedicated function which cannot be changed; 2) **high speed I/O** - high speed inputs or outputs with processing times of <200ns that have certain functions associated to them, but can also be used as general purpose inputs or outputs; and 3) **general purpose I/O**, which have no built-in associated functions and are defined by the user.

There are also two analog inputs and two PWM outputs.

2.1.3.1 Dedicated Input/Outputs

The Commander core includes a number of inputs and outputs that have dedicated functions. These include safety-related functions such as **emergency stop** and **limit switches**, informational inputs such as **slowdown** and **home**, and functional inputs such as **external start** and **(MPG) pulsar inputs**.

2.1.3.1.1 Mechanical input signals

The following four signals can be input for each axis:

1) +L (Limit +) When this signal turns ON while operating in the positive direction, the motor stops immediately, or decelerates and stops.
2) -L (Limit -) When this signal turns ON while operating in the negative direction, the motor stops immediately, or decelerates and stops.
3) SD (Slowdown) Used as a deceleration signal or a deceleration stop signal by a software setting.
4) H (Home) Input signal for a Homing operation.

The input logic for these signals can be changed with software.

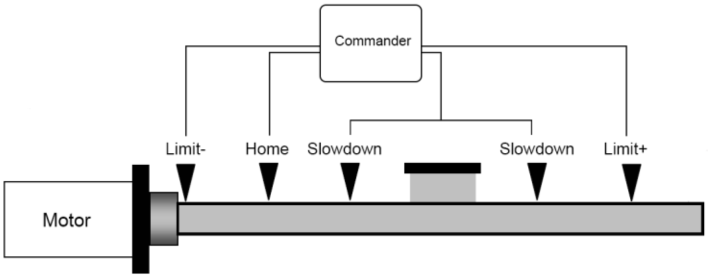


Figure 2-7

2.1.3.1.1.1 Limit Switches

Limit switches, most commonly seen in linear mechanisms to restrict motion to a safe range, are used to identify that the motor has reached its end of travel. If either the positive (**L+**) or negative (**L-**) signal becomes active, the Commander core is instructed to immediately stop all advancement in that direction but will allow movement in the opposite direction. As such it is important to ensure the placement of the positive limit at the end of positive travel and negative limit at the end of negative travel. The polarity of limit inputs is set with the **POL[axis](Bit3).**

When designing your limit circuit, here are a few points to keep in mind:

1) It is especially important that the motor must stop when the signal is active.
2) To prevent nuisance limit faults that can stop your system, ensure that electrical noise does not trigger your limits as active.
3) To operate in a fail-safe manner, the signal line must be designed to trigger as active when the cable connecting the limit switch to the Commander core becomes disconnected.

With the above points in mind think about your limit circuit and the active logic level in your system.

Limit error

By default **IERR=0,** which will trigger a limit error **MST[axis](bit8~9)** when a limit switch is set to active. Once the limit error is set, use the **CLR[axis]** command to clear the error in ASCII mode or the **ECLEAR[axis]** command in standalone mode.

When using the end limit during the homing modes, the **ERC** (deflection counter clear) signal can be setup to clear the limit errors that accrue due to homing automatically.

The limit error states can be ignored by setting **IERR=1**. In this case, the motor will still stop when the appropriate switch is triggered; however, it will not enter an error state.

ASCII	CLR	IERR	ERC[axis]	MST[axis]
Standalone	ECLEAR[axis]	—	—	MST[axis]

2.1.3.1.1.2 Slowdown switches

The slowdown switch (**SD**) input indicates a point to begin deceleration. If the slowdown switch inputs are active when the motor is operating at HSPD speed and this signal is triggered, the motor will start to decelerate.

The 2-bit **SDE[axis]** command enables the SD inputs and sets the latching function. By default, **SDEX=0** disables the X-axis SD inputs. The inputs can be enabled without a latch function using **SDEX=1**, or enabled with a latch function **SDEX=3**.

Bit	Description	Setting	
0	SD input	0 - Disabled	1 - Enabled
1	SD latching	0 - Disabled	1 - Enabled

Table 2-1 SDE command

The **SDC[axis]** command determines the specific operation of the SD input. By default, **SDC[axis]=0** sets the operation, when triggered, to decelerate the motor to LSPD without stopping. When **SDC[axis]=1,** the operation is set to decelerate and stop when the SD input is triggered.

Value	Description
0	Decelerate only
1	Decelerate and stop

Table 2-2 SDC command

Triggering the SD input when **SDC=1** will set the SD stop flag, **MST[axis](bit17)**. The SD stop flag will not prevent further motion, but can only be cleared by using the **CLR[axis]** command in ASCII mode or the **ECLEAR[axis]** command in standalone mode.

The SD input, when triggered, will cause the motor to slowdown regardless of the direction of travel.

Figure 2-7 shows an example of a setup using the slowdown inputs. These inputs are mainly used in homing operations. In this case the axis would move at HSPD, trigger the slowdown input, allowing the motor to decelerate to LSPD, before it triggers the home input.

Other uses for the slowdown input are placing SD limits at each end of travel just before the end limits. This allows for two options. The slowdown inputs function as a soft limit (**SDC=1**) that decelerates and stops the motion, allowing the end limits to operate as emergency hard end limits. Or they can just allow the motor to decelerate (**SDC=0**) to LSPD before triggering the end limits. The polarity of the slowdown inputs is set with the **POL[axis](Bit6)**.

ASCII	SDE	SDC	POL[axis]	CLR	MST[axis]
Standalone	—	—	—	ECLEAR[axis]	MST[axis]

2.1.3.1.1.3 Home switch

The Home (**H**) switch is used during some homing modes to aid in the detection and location of the zero-position used in **absolute** positioning operations. For more information see homing operation in section 2.2.5 Homing operation and **Appendix A: Homing Mode Actions**. The polarity of the home input is set with the **POL[axis](Bit4)**.

2.1.3.1.2 Emergency stop

The Commander core has an **emergency stop (/CEMG)** input for dedicated use as an emergency stop signal. When in operation, if the **/CEMG** input goes low or the **ESTOP** command is sent, all axes will stop immediately. No axis can operate while the /CEMG signal is low.

The logical input of the /CEMG input cannot be changed.

Note: In a normal stop operation, the final pulse width is normal. However, in an emergency stop operation, the final pulse width may not be normal. As such the motor drivers may not recognize the last pulses, and therefore the Commander core internal counter may not match the mechanical position. Therefore, after an emergency stop, it is recommended that you perform a homing operation to match the command and mechanical positions.

Emergency stop error

No axis can operate while the **/CEMG** signal is low.

By default when **IERR=0,** any axis that is operating when the **/CEMG** signal goes low or the **ESTOP** command is sent will set an **EMG** error flag **MST[axis](bit16)** for that axis. In addition, any axis that tries to start operating while the **/CEMG** signal is low will set an **EMG** error flag for that axis. Any axis with a set **EMG** error flag cannot operate until the **EMG** error flag has been cleared. However, if an axis was not moving when the **/CEMG** or **ESTOP** was triggered, the **EMG** error flag will not be set for that axis, and thus that axis can operate immediately after the **/CEMG** input is cleared.

To clear the **EMG** error flag, use the **CLR[axis]** command in ASCII mode or the **ECLEAR[axis]** command in standalone mode.

The **EMG** error flags are ignored by setting **IERR=1**. In this case, all axes will still stop when **/CEMG** is triggered or **ESTOP** command is sent; and all axes can operate again immediately after the **/CEMG** input is cleared.

ASCII	ESTOP	MST[axis]	CLR	IERR
Standalone	—	MST[axis]	ECLEAR[axis]	—

2.1.3.1.3 External Start

The **external start (/CSTA)** input allows motion to be started by taking the **/CSTA** input low.

By default, the external start input is disabled (**EXST=0),** which will allow motion to start immediately upon receipt of any motion command. If the external start input is enabled (**EXST=1**), then any motion command sent will wait for the **/CSTA** input to go low before motion is started. The Commander core can buffer up to 3 motions, on a first-in first-out order. They will start one at a time each time the **/CSTA** input goes low.

The logical input of the **/CSTA** input cannot be changed.

ASCII	EXST
Standalone	—

2.1.3.1.4 Manual pulse generator (MPG) inputs

An MPG is similar to an encoder in that it is a type of pulse generator that can be used to communicate to the Commander core a speed and direction. The difference between the two is that an MPG commands motion, whereas an encoder reports the motion.

The MPG input can be used for a number of functions such as:

1) A manual pulse generator, which allows for fine adjustment of an axis.

2) Cam follower, which will allow the axis to follow a cam axis based on its encoder output.
3) Electronic gearing, which allows an axis to follow the input commands but at a different rate of speed (faster or slower).

The Commander core has a dedicated MPG input for each axis. To enable the MPG input, set the **MPE[axis]** command to 1. To disable the MPG input, set the **MPE[axis]** command to 0.

The MPG inputs are controlled by three commands: MPG signal input mode (**POL[axis]**), multiplication (**MPM[axis]**) and division (**MPD[axis]**). These work together to set how the pulses from the MPG input controls the motion of an axis.

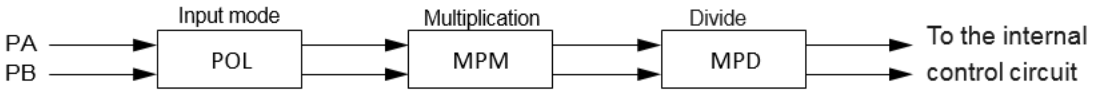


Figure 2-8

MPG signal input mode

The Commander core can accept two different pulse types which can be set using the **POL[axis](Bit15~16)**.

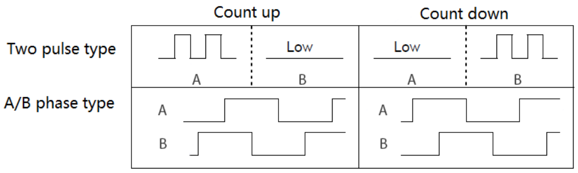


Figure 2-11

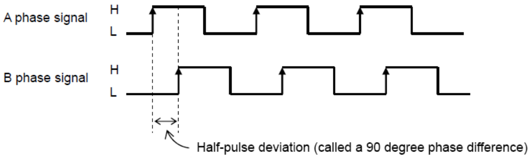


Figure 2-11

1) Two pulse type (not common) is where pulses are sent to A input to count up and B input to count down. Each pulse input is one count of the encoder.
2) A/B phase type (common).
   a. For an A/B phase type there should be 90° difference between the pulses from A and B. See Figure 2-11.
   b. By default, when the A phase leads the B phase motion is in the positive direction, when B phase leads the A phase motion is in the negative direction. The direction can be changed by **POL[axis](Bit14)**. See Figure 2-11.
   c. For an A/B phase type you can select x1, x2 or x4 multiplication rate. Figure 2-12 shows how the A/B phase counts would be interpolated for a x1, x2, and x4 multiplication rate.

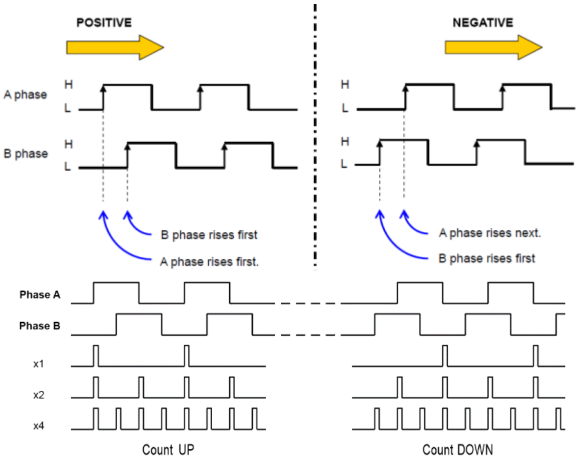


Figure 2-12

      i. x1 multiplication – Only the rising edge of the A phase is counted, B phase is not counted and only used to check the direction of motion.
      ii. x2 multiplication – The rising and falling edge of the A phase is counted, B phase is not counted and only used to check the direction of motion.
      iii. x4 multiplication – The rising and falling edge of the A and B phase is counted, and used to check the direction of motion.

Maximum frequency input on the MPG line is the maximum pulses that can be input to the MPG input, not the counts. Since most manual pulse generator outputs are rated based on a x4 multiplication rate, the output frequency of pulses is most likely ¼ the count rate output. Please check the datasheet for your manual pulse generator to confirm.

Electronic gearing the MPG signal

After the input mode stage, the MPG pulse signal is converted to counts based on the **POL[axis]** settings. The counts then pass through an electronic gearing stage, which includes a multiplier **MPM[axis]** and divider **MPD[axis]** circuit.

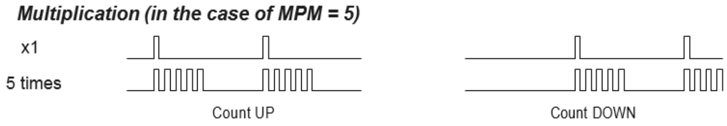


Figure 2-13

MPM[axis] can be used to access the multiplier. The multiplier can be set to a value from [1 to 32]. A value of 1 will turn off the multiplier.

MPD[axis] is the numerator of the divider circuit. The **MPD[axis]** can be set to a value from [1 to 2048]. The denominator of the divider circuit is fixed as 2048. A value of 2048 will turn the divider off.

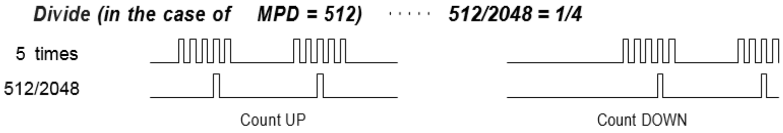


Figure 2-14

The multiplier and divider circuits can be used together to create any ratio. The multiplier and divider use the following equation to determine the number of output counts[2]:

$$\text{Output counts} = (\text{MPG counts}) * \frac{\text{MPM} * \text{MPD}}{2048}$$

MP[axis] can be written to set an MPG position or read to see the current position. The counter value here is the final counts after the electronic gearing stage.

ASCII	MPD[axis]	MPE[axis]	MPM[axis]	MP[axis]	POL[axis]
Standalone	—	MPGE[axis]	—	—	—

2.1.3.2 High-speed Inputs/Outputs

The Commander core has a number of high-speed inputs and outputs that have dedicated functions. These include **position latching inputs** and **synchronization outputs.** If these functions are not needed for your application, they can also be used as **general-purpose inputs or outputs**.

2.1.3.2.1 High-Speed Digital Inputs

The Commander core provides four (4) high-speed inputs. Each input can independently be set as a **position latching input** or **general-purpose input.** By default, these inputs are set to general-purpose input.

Position Latching Inputs

Any of the four high-speed inputs can be set independently to be position latch inputs by enabling the feature using the **LT[axis]** command for the corresponding axis. This feature will allow the controller to perform a high-speed position capture of both pulse and encoder counters based on a digital input trigger. The timing between digital input trigger being received and the capture of the pulse and encoder position is 150 to 200ns.

Each axis has a designated digital input to perform a high-speed position latch. See corresponding latch input for each axis in Table2- 3 latching inputs below.

[2] MPG counts are counts after the input mode setup done by POL[axis].

Axis	Input
X	DI1/LTCx
Y	DI2/LTCy
Z	DI3/LTCz
U	DI4/LTCu

Table2- 3 latching inputs

Use the **LT[axis]** command to enable and disable the latch feature. To read the latch status, use the latch status **LTS[axis]** command. Table 2-4 LTS command return below describes the possible return values of the latch status command.

Value	Condition
0	Latch off
1	Latch on and waiting for latch trigger
2	Latch triggered

Table 2-4 LTS command return

Once the latch is triggered, the triggered pulse position can be retrieved using **LTP[axis]** command. Similarly, the triggered encoder position can be retrieved using the **LTE[axis]** command.

When StepNLoop mode is enabled, the position value is invalid.

ASCII	LT[axis]	LTS[axis]	LTP[axis]	LTE[axis]
Standalone	LT[axis]	LTS[axis]	LTP[axis]	LTE[axis]

High-Speed Digital Inputs

Any of the four high-speed inputs can be set independently to be general-purpose inputs by disabling the position latch feature using the **LT[axis]** command for the corresponding axis. These inputs are set to general-purpose by default.

The digital input status of all four available inputs can be read with the **DI** command. Digital input values can also be referenced one bit at a time by using the **DI[1-4]** commands. Note that the indexes are 1-based for the bit references. For example, DI1 refers to bit 0, not bit 1. See Table 2-5 DI command below for details.

Bit	Input	Bit-Wise Command
0	Digital Input 1	DI1
1	Digital Input 2	DI2
2	Digital Input 3	DI3
3	Digital Input 4	DI4

Table 2-5 DI command

If a digital input is on, the corresponding bit of the **DI** command is 1. Otherwise, the bit status is 0. The voltage level required to activate a digital input is determined by the polarity setting. The **DIP** command can be used to toggle the polarity of the digital inputs.

ASCII	DI	DI[1-4]	DIP	LT[axis]
Standalone	DI	DI[1-4]	—	LT[axis]

2.1.3.2.2 High-Speed Digital Output

The Commander core provides four (4) high-speed outputs. Each output can independently be set as a **synchronization output** or a **general-purpose output.** By default, the outputs are set to general-purpose outputs.

Synchronization Outputs

The Commander core provides four high-speed synchronization outputs. This feature allows the controller to perform a high-speed comparison between the encoder or pulse **position counter** and a **set condition (comparator)**, which is set with the **SYNP[axis]** command. When the condition is met, the corresponding synchronization output is enabled. The timing between the condition being met and the synchronization output being triggered is 150 to 200ns.

There are three modes for the synchronization outputs to be enabled, which are set with the **SYNC[axis]** command:

1. **At position mode** (counter = comparator)
2. **> or < position mode** (counter is either > or < then the comparator)
3. **Continuous mode** (every comparator position)

Each axis has a designated high-speed synchronization output. While synchronization output is enabled for an axis, the corresponding digital output cannot be manually controlled by the user. See corresponding synchronization output for each axis in Table 2-6 synchronization output below.

Axis	Output
X	DO1/SYNCx
Y	DO2/SYNCy
Z	DO3/SYNCz
U	DO4/SYNCu

Table 2-6 synchronization output

Use the **SYNO[axis]** command to enable and the **SYNF[axis]** command to disable the synchronization feature. To read the synchronization status, use the latch status **SYNS[axis]** command.

Value	Condition
0	Synchronization output configuration is disabled
1	Wait for the establishment of comparison condition
2	Comparison condition is established

Table 2-7 SYNS command return

In **continuous mode**, there is an option for a synchronization window function. When enabled with the **SYNWO[axis]** command, the synchronization function will run continuously, but only outputs a pulse when the counter is between the synchronization maximum **SYNMAX[axis]** and minimum **SYNMIN[axis]** points. There is no loss of synchronization when outside the synchronization window while continuous mode is enabled. To disable the window function use the **SYNWF[axis]** command, synchronization will not be lost. To simultaneously disable the window function and continuous sync mode use the **SYNF[axis]** command.

The **DOP** can be used to toggle the polarity of the digital outputs.

ASCII	SYNO[axis]	SYNWO[axis]	SYNF[axis]	SYNWF[axis]	SYNS[axis]
Standalone	SYNON[axis]	—	SYNOFF[axis]	—	SYNSTAT[axis]

ASCII	SYNP[axis]	SYNC[axis]	SYNMAX[axis]	SYNMIN[axis]	DOP
Standalone	SYNPOS[axis]	SYNCFG[axis]	—	—	—

High-Speed Digital Outputs

Any of the four high-speed outputs can be set independently to be general-purpose outputs by disabling the synchronization feature using the **SYNF[axis]** command for the corresponding axis. These outputs are set to general-purpose by default.

The 4-bit **DO** command can be used to set all the digital outputs at once. The **DO** value is a decimal number and must be within the range of 0-15.

Digital outputs can also be set one at a time with the **DO[1-4]** commands. Note that the indexes are 1-based for the bit references. For example, DO1 refers to bit 0, not bit 1. See Table 2-8 DO command below for details.

Bit	Output	Bit-Wise Command
0	Digital Output 1	DO1
1	Digital Output 2	DO2
2	Digital Output 3	DO3
3	Digital Output 4	DO4

Table 2-8 DO command

If a digital output is turned on, the corresponding bit of the **DO** command is 1. Otherwise, the bit status is 0. The voltage level of the digital output when it is on or off is determined by the polarity setting. The **DOP** command can be used to toggle the polarity of the digital outputs.

The initial state of the digital outputs can be defined by setting the 4-bit **DOBOOT** register to the desired initial digital output value. The **DOBOOT** value must be within the range of 0-15. The value is stored to flash memory once the **STORE** command is issued.

Bit	Output	Setting
0	Digital Output 1	0 - OFF 1 - ON
1	Digital Output 2	0 - OFF 1 - ON
2	Digital Output 3	0 - OFF 1 - ON
3	Digital Output 4	0 - OFF 1 - ON

Table 2-9 DOBOOT command

ASCII	DO	DO[1-4]	DOP	DOBOOT	SYNF[axis]	STORE
Standalone	DO	DO[1-4]	—	—	SYNOFF[axis]	STORE

2.1.3.3 General-purpose input/output

The Commander core has a number of general-purpose inputs and outputs: **configurable digital I/O, analog inputs** and **PWM outputs.** There is a built-in function that will allow a **joystick** connected to the analog inputs to be used for control of the X and Y-axis.

2.1.3.3.1 Configurable Digital I/O

The Commander core has 32 general-purpose configurable digital I/O available. The **IO** command can be used to access the current configurable I/O status. The value must be a 32-bit integer.

Configurable digital I/O can also be accessed one bit at a time using the **IO[1-32]** commands. Note that the indexes are 1-based for the bit references. For example, IO1 refers to bit 0, not bit 1.

In order to set a configurable digital I/O as an input or output, use the **IOCFG** command. If a bit in the 32-bit **IOCFG** value is set to 0, the corresponding configurable digital I/O will be defined as an input. If the bit is 1, the configurable digital I/O will be defined as an output.

The initial state of the configurable digital I/O can be defined by setting the 32-bit **IOBOOT** register to the desired initial configured output value. The **IOBOOT** value will only be applicable to configurable I/O defined as outputs by the **IOCFG** command. The **IOBOOT** value must be a 32-bit value. The value is stored to flash memory once the **STORE** command is issued. The settings take effect on the next power cycle.

The 2-bit **IOP** command can be used to toggle the polarity of the configurable I/O. See Table 2-10 IOP command below.

Bit	Description	Setting	
0	Logic of input setting	0 - Negative logic Active Low	1 - Positive logic Active High
1	Logic of output setting	0 - Negative logic Active Low	1 - Positive logic Active High

Table 2-10 IOP command

ASCII	IO	IO[1-32]	IOBOOT	IOCFG	IOP	STORE
Standalone	IO	IO[1-32]	—	—	—	STORE

2.1.3.3.2 Analog Input/output

The Commander core has two 10-bit analog inputs available. The **AI[1-2]** command can be used to read the current analog input value. The return value is in milliVolts and can range from 0 to 3300.

The voltage supplied to the analog inputs should stay within the 0V to 3.3V range.

ASCII	AI[1-2]
Standalone	AI[1-2]

Joystick Control

Joystick control is available on the Commander core for the X and Y-axis. When this mode is enabled, the pulse speed and direction outputs for the X and Y axes can be controlled by the corresponding analog input. See the axis to analog input assignment in the table below.

Axis	Input
X	AI1
Y	AI2

Table 2-11 analog inputs

To enable or disable joystick control for an axis, use the ASCII command **JENA** or the standalone command **JOYENA**. The joystick enable parameter is a 2-bit value. For example, A joystick enable value of 3 means the joystick feature is enabled on both the X and Y-axis. If joystick control is enabled, StepNLoop is automatically disabled.

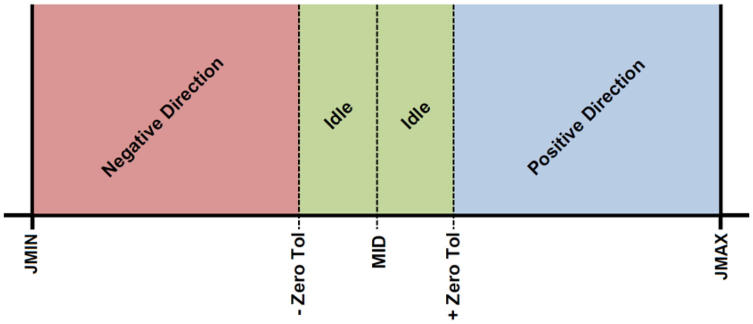


Figure 2-15

For the Commander core to properly translate the analog inputs into a safe pulse speed and direction output for your system, you first must define what is acceptable and safe for your system. This includes defining the relation between voltage and **direction** of travel, and the zero-tolerance (or dead) zone during which there is no output from the Commander core. When the input voltage is within the voltage (**± JTOL**) range before and after the midpoint (**MID**) between the maximum input voltage (**JMAX**) and the minimum input voltage (**JMIN**), the movement of the axis is stopped. See Figure 2-15.

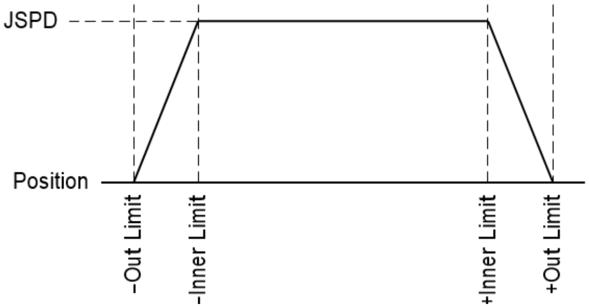


Figure 2-16

The **safe speeds** are defined as the maximum speed the analog input can command and maximum rate of speed change. The joystick **positioning** soft limits define where the analog inputs can command a system to move. Soft limit values can be set for the inner and outer limits, in the positive and negative direction for both axes (**JLIM**). When it goes beyond the inner limit, it decelerates and stops, and if it goes beyond the outer limit, it immediately stops. See Figure 2-16.

For more information on setting the limits refer to Section 2.5.1 Joystick operation and the individual commands in the Command Reference.

ASCII	JENA	JMAX[axis]]	JMIN[axis]	JSPD[axis]	JDEL[axis]	JTOL[axis]	JLIM[number]
Standalone	JOYENA	—	—	JOYHS[axis]	JOYDEL[axis]	—	—

2.1.3.3.3 Pulse-width modulated Outputs.

The Commander core has two pulse-width modulated (PWM) outputs available. The PWM outputs are 500Hz and can range from a 1% to 99% on-time duty cycle. The **PWM[1-2]** command is used to set the value which is in percent and can range from 1 to 99.

PWM1 is % of the time the signal is low, and PWM2 is % of time the signal is high.

ASCII	PWM[1-2]
Standalone	PWM[1-2]

2.1.4 Motion Profile

Any given motion profile can be plotted as a trapezoid, where Time is the base, Velocity is the height, Acceleration rate is the angle and Distance is the area.

By default, the Commander core uses a linear acceleration profile as shown in Figure 2-17.

To reduce jerk, an S-curve acceleration profile as shown in Figure 2-18 can also be achieved by using the **SCV[axis]** command. Setting this command to 1 will enable s-curve for the indicated axis.

The S-curve acceleration profile reduces jerk, but it requires more power from the motor than a linear acceleration profile, when the time for acceleration is the same.

For a typical move, the axis will start moving at the low-speed setting and accelerate to the high-speed setting. Once at high-speed, the motor will move at a constant speed until it decelerates from high-speed to low-speed and immediately stops.

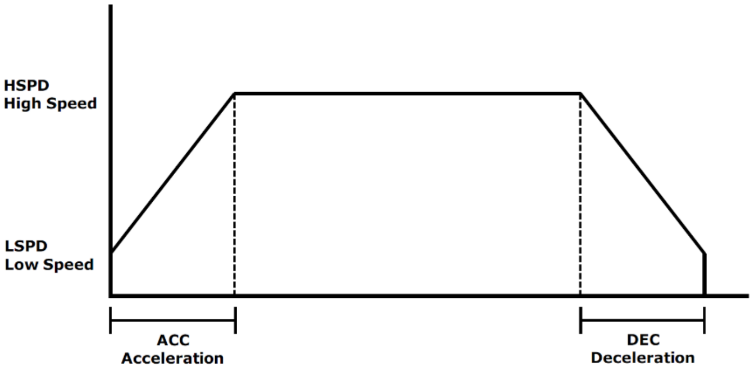


Figure 2-17 Linear acceleration

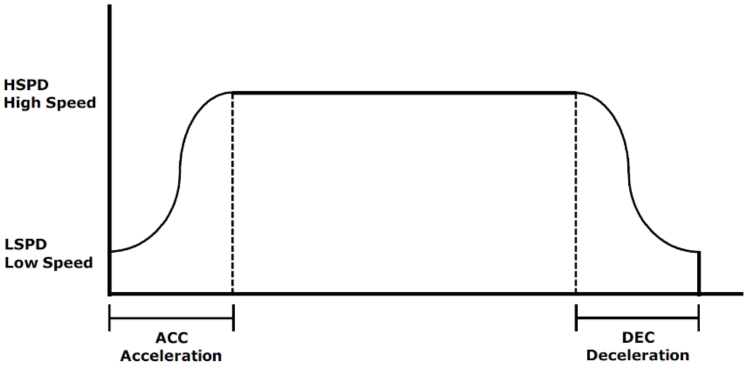


Figure 2-18 S-curve acceleration

High-speed and low-speed are in pps (pulses/second). Use the commands **HSPD[axis]** and **LSPD[axis]** to set/get individual high speed and low-speed settings in both ASCII and standalone mode. To access the global high-speed and low-speed values, use the ASCII/standalone commands **HSPD** and **LSPD**.

Acceleration and deceleration time are in milliseconds. Use the **ACC[axis]**/**DEC[axis]** command to set/get individual acceleration/deceleration values. To access the global acceleration/deceleration value, use the corresponding **ACC** or **DEC** command.

The minimum and maximum acceleration values depend on the high-speed and low-speed settings. Refer to Appendix B: Speed Settings for details.

Individual speed and acceleration settings will take priority over global settings. A global setting will only be used under the following conditions:

1) The corresponding individual setting is defined as zero.
2) Coordinated motion is being performed.

ASCII	HSPD	LSPD	ACC	DEC	HSPD[axis]	LSPD[axis]	ACC[axis]	DEC[axis]	SCV[axis]
Standalone	HSPD	LSPD	ACC	DEC	HSPD[axis]	LSPD[axis]	ACC[axis]	DEC[axis]	—

2.1.5 Power-on settings

The Commander core has the ability to set and store the default power-on value for the digital outputs and the stand-alone programs.

The initial state of the enable outputs can be defined by setting the 4-bit **EOBOOT** register to the desired initial enable output value.

The initial state of the high-speed digital outputs can be defined by setting the 4-bit **DOBOOT** register to the desired initial digital output value. The **DOBOOT** value must be within the range of 0-15.

The initial state of the configurable digital I/O can be defined by setting the **IOBOOT** register to the desired initial configured output value. The **IOBOOT** value will only be applicable to configurable I/O defined as outputs by the **IOCFG** command. The **IOBOOT** value must be a 32-bit value.

Standalone programs can be configured to run on boot-up using the **SLOAD** command.

The values for the power-on settings are stored to flash memory once the **STORE** command is issued. They will take effect on the following power cycle.

ASCII	EOBOOT	DOBOOT	IOBOOT	IOCFG	SLOAD	STORE
Standalone	—	—	—	—	—	STORE

2.1.6 System settings

The Commander core has the ability to set and store the following system settings:

Command	Description
ERC[axis]	Setting the conditions for output of the ERC (error clear) signal.
ERCD[axis]	Setting the delay time of the ERC signal.
ERCP[axis]	Setting the on-time of the ERC signal.
EXST	Enables the use of the external start signal.
IERR	Disables the error flag for the end-limit and alarm signal.
INP	Enables the use of the INP signal.
POL[axis]	Sets the input modes and the polarity of the pulse, direction outputs, end limits, home, alarm, SD, INP, ERC, EO, encoder, index, and MPG signals.
SCV	Enables the use of S-curve acceleration profile
SDC	Sets the operation when the SD signal is detected.
SDE	Enables the use of the SD signal.
STORE	Saves the settings of commands to the flash memory.

Table 2-12 system settings

2.1.6.1 ERC (deflection counter clear)

The Commander core can be setup to send an **ERC** (deflection counter clear) signal to clear any external errors from the servo drive. The ERC signal is configured with the following commands:

ERC[axis] command is 2-bit and sets the output condition of the ERC signal. The **ERC[axis]** command can write a new condition or read the condition based on the chart below.

Bit	Condition for Output	Setting	
0	Error Stop	0: Disabled (Not used)	1: Enabled (Used)
1	Homing operation complete	0: Disabled (Not used)	1: Enabled (Used)

Table 2-13 ERC command

ERCD[axis] command sets the delay time used when a deviation clear signal is sent. The **ERCD[axis]** command can write a new delay or read the condition based on Table 2-14 ERCD command below.

Value	Delay
0	0 µs
1	12 µs
2	1.6 ms
3	104 ms

Table 2-14 ERCD command

ERCP[axis] command sets the width of the output deviation clear signal. The **ERCP[axis]** command can write a new pulse width or read the condition based on Table 2-15 ERCP command below.

Value	Pulse Width
0	12 µs
1	102 µs
2	409 µs
3	1.6 ms
4	13 ms
5	52 ms
6	104 ms

Table 2-15 ERCP command

ASCII	ERC	ERCD	ERCP
Standalone	—	—	—

2.1.6.2 INP (in-position)

The **INP[axis]** command sets if the Commander core will use the **INP** (in-position) signal to determine if the motor has reached the requested position. When the **INP[axis]** function is enabled the operation completion status is delayed until an in-position signal is received. The **ERCP[axis]** command can write a new pulse width or read the condition based on the chart below.

Value	Discription
0	Disabled (Not used)
1	Enabled (Used)

Table 2-16 INP command

ASCII	EXST
Standalone	—

2.1.6.3 Input modes and the polarity of the input/output logic

The POL[axis] command is a 17-bit value used to set or read the input mode and/or input/output logic of various signals. Use Table 2-17 POL command below to see description of each bit setting.

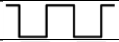
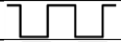
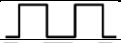
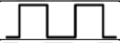
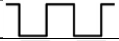
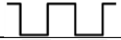
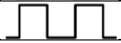
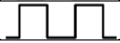
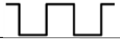
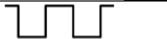
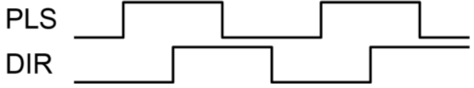
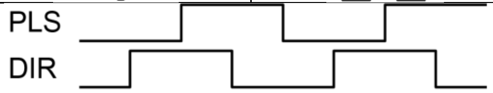
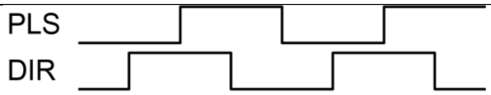
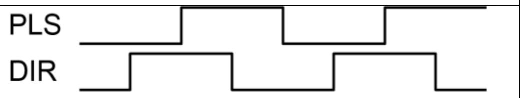
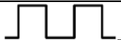
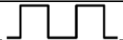
Bit	Description			
0~2	The output modes of command pulse signals			
	Positive direction operation		Negative direction operation	
	PUL output	DIR output	PUL output	DIR output
000		High		Low
001		High		Low
010		Low		High
011		Low		High
100		High	High	
101				
110				
111		Low	Low	
Bit	**Description**		**Setting**	
3	Logic of End limit signal (+/-L)		0 Positive logic	1 Negative logic
4	Logic of home signal (H)		0 Negative logic	1 Positive logic
5	Logic of alarm signal (ALM)		0 Negative logic	1 Positive logic
6	Logic of deceleration signal (SD)		0 Negative logic	1 Positive logic
7	Logic of in-position (INP)		0 Negative logic	1 Positive logic
8	Logic of deviation counter clear signal (ERC)		0 Negative logic	1 Positive logic
9	Logic of enable signal (EO)		0 Negative logic	1 Positive logic
10	Encoder input direction		0 Do Not Reverse	1 Reverse
11~12	Specification of the feedback pulse signal		00 x1	01 x2
			10 x4	11 two pulse
13	Logic of Z-index signal		0 Falling Edge	1 Rising Edge
14	MPG input direction		0 Do Not Reverse	1 Reverse
15~16	Specification of the MPG input signal		00 x1	01 x2
			10 x4	11 two pulse

Table 2-17 POL command

The polarity can also be set for the digital inputs, digital outputs, and configurable I/O. The **DIP** command can be used to toggle the polarity of the digital inputs and the **DOP** can be used to toggle the polarity of the digital outputs. For these 0 = Negative logic, 1 = Positive logic.

The **IOP** command is 2-bit and can be used to toggle the polarity of the configurable I/O. Bit 0 sets the input logic and Bit 1 sets output logic. See Table 2-18 IOP command below.

Bit	Description	Setting	
0	Logic of input setting	0 Negative logic	1 Positive logic
1	Logic of output setting	0 Negative logic	1 Positive logic

Table 2-18 IOP command

ASCII	POL[axis]	DIP	DOP	IOP
Standalone	—	—	—	—

2.1.7 Flash Memory

The Commander core has a built-in flash memory that allows it to store many of the system settings that are loaded on startup.

Once set, using the **STORE** command will save the settings in the table below to Flash memory.

DB	ERC[axis]	INP[axis]	JLIM[number]	MPD[axis]	SDC[axis]	SLOAD[prg]
DIP	ERCD[axis]	IOBOOT	JMAX[axis]	MPE[axis]	SDE[axis]	SLR[axis]
DOBOOT	ERCP[axis]	IOCFG	JMIN[axis]	MPM[axis]	SL[axis]	SLT[axis]
DOP	EXST	IOP	JSPD[axis]	POL[axis]	SLA[axis]	V[0-99]
EINT	ID	JDEL[axis]	JTOL[axis]	SAP	SLE[axis]	ZCNT
EOBOOT	IERR	JENA				

Table 2-19 STORE command

2.2 Single-axis and Multi-axis operations

Singe-axis operations involve the moving of a single axis and do not require interaction with any other axis. **Multi-axis** operations involve the moving of two or more axes; however, it does not require interaction between the axes. This section will cover basic function blocks that are used for these operations.

2.2.1 Motor Power

The enable output (**EO**) command is most often used to command the driver to enable or disable the current to the motor. The effect of the enable output signals will depend on the characteristics of the motor drive. The polarity of this output needs to match the drive and is set with the **POL[axis](Bit9)**.

The enable output (**EO**) command is a 4bit value where a bit value of 1 is enabled and 0 is disabled for the corresponding axis. **EO** has a range of 0-15.

Enable output values can also be referenced one bit at a time by the **EO[1-4]** commands. Note that the indexes are 1-based for the bit references (i.e. EO1 refers to bit 0, not bit 1).

Bit	Output	Bit-Wise Command
0	EOx [X-axis]	EO1
1	EOy [Y-axis]	EO2
2	EOz [Z-axis]	EO3
3	EOu [U-axis]	EO4

Table 2-20 EO command

The initial state of the enable outputs can be defined by setting the 4-bit **EOBOOT** register to the desired initial enable output value. The value is stored to flash memory once the **STORE** command is issued.

Bit	Output	Setting	
0	EOx [X-axis]	0 - Disabled	1 - Enabled
1	EOy [Y-axis]	0 - Disabled	1 - Enabled
2	EOz [Z-axis]	0 - Disabled	1 - Enabled
3	EOu [U-axis]	0 - Disabled	1 - Enabled

Table 2-21 EOBOOT command

ASCII	EO	EO[1-4]	EOBOOT	STORE	POL[axis]
Standalone	EO	EO[1-4]	—	STORE	—

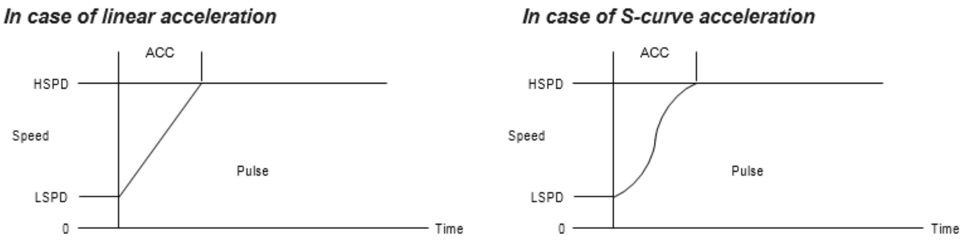
2.2.2 Jog operation

A jog move is used to continuously move the motor without stopping. Use the **J[axis]+**/**J[axis]-** command when operating in ASCII mode and the **JOG[axis]+**/**JOG[axis]-** in standalone mode.

The Commander core will output pulses to move the motor immediately upon receipt of the command. Any number of motors can be moved independently at the same time.

The motor will start moving at the defined **low speed**, then will accelerate as defined by **the acceleration time and type**, to the final defined **high speed**.

Once this move is started, the speed can be adjusted by using the **on-the-fly speed change** command; however, the motor will only stop if a limit input is activated during the move or a **STOP** command is issued.



In case of linear acceleration

In case of S-curve acceleration

Figure 2-19

If a motion command is sent while the controller is already moving, the command is not processed. Instead, an error response is returned. See Table 4-41 in 4.2 Error Codes for details on error responses.

ASCII	J[axis][+/-]
Standalone	JOG[axis][+/-]

Operating procedure

If there is no change in the previous set data, the operation 1 below is not necessary.

1) Set the operating speeds
2) Issue the **J** command
3) Issue stop command

Operating conditions

- Joystick operation is disabled
- MPG operation is disabled
- Operation is stopped
- The error status is clear

Related commands:

LSPD, HSPD, ACC, DEC, J, SCV, STOP, ABORT, ESTOP, MST

2.2.3 Stopping

When the motor is performing any type of move, motion can be stopped abruptly or with deceleration. It is recommended to use decelerated stops so that there is less impact on the system. The **ABORT[axis]** command will immediately stop an individual axis. Use the **ABORT** command to immediately stop ALL axes.

To employ deceleration on a stop, use the **STOP[axis]** command to stop an individual axis. Use the **STOP** command to stop ALL axes.

If an interpolation operation is in progress when a **STOP[axis]** or **ABORT[axis]** command is entered, all axes involved in the interpolation operation will stop.

ASCII	ABORT	ABORT[axis]	STOP	STOP[axis]
Standalone	ABORT	ABORT[axis]	STOP	STOP[axis]

2.2.4 Positioning operation

Unlike a **Jog** operation, a **Positioning** operation is a move with a specified stop point. The stop point will be either an **absolute** position in relation to the 0 start point that was defined during **homing**, or an **incremental** position from the motor's current position.

The motor will start moving at the defined **low speed**, then will accelerate based on **the acceleration time and type**, to the final defined **high speed**. Once it gets close to the stop point, it will decelerate as defined by the deceleration **time and type** to the defined **low speed**, then the motor will stop**.**

At any time while the motor is moving, the specified stop point can be adjusted by using the **on-the-fly target position change** command, and the speed it is moving can be adjusted by using the **on-the-fly speed change** command.

In addition, the motor will stop without completing the defined positioning operation if a limit input is activated during the move or a **STOP**/**ABORT** command is issued.

The Commander core can perform positional moves for individual axes, multiple axes, or **linear coordinated** motion.

2.2.4.1 Absolute or Incremental

The Commander core can perform positional moves in absolute or incremental mode. For absolute mode, the **ABS** command is used, and for incremental mode the **INC** command is used. Best practice is to set the absolute or incremental mode during system setup. The move mode will remain in absolute or incremental mode until it is changed.

In absolute mode, the axis will move to the specified target position. In incremental mode, the axis will increase or decrease its current position by the specified target position.

The **MM** command can be used to determine the current move mode of the controller. Table 2-22 MM command below details the value assignment of the **MM** command.

Value	Setting
0	Absolute Mode
1	Incremental Mode

Table 2-22 MM command

ASCII	ABS	INC	MM
Standalone	ABS	INC	—

Operating procedure

If there is no change in the previous set data, the operation 1 and 2 below is not necessary.

1) Set the operating speeds
2) Set the positioning mode with **ABS** or **INC** command
3) Issue a positioning command

Operating conditions

- Joystick motion is disabled
- MPG operation is disabled
- Operation is stopped
- The error status is clear

Related commands

LSPD, HSPD, ACC, DEC, X, Y, Z, U, SCV, ABS, INC, MST

2.2.4.2 Individual Position Moves

For individual axis control use **X**, **Y**, **Z** and **U** commands followed by the target position value. For example, the **X1000** command will move the X-axis to position 1000 if performed in absolute mode. **X1000Y2000** command will move the X-axis to position 1000 and Y-axis to position 2000 if performed in absolute mode. Any number of axes can be moved independently at the same time. If **EINT=0** then the axes will move independently; however, if **EINT=1** the move will be linearly interpolated. For more information see section 2.3.1 Linear interpolation.

The Commander core will output pulses to move the motor immediately upon receipt of the command.

ASCII	X[target]	Y[target]	Z[target]	U[target]	EINT
Standalone	X[target]	Y[target]	Z[target]	U[target]	EINT

2.2.5 Homing operation

Home search routines involve moving the motor and using the home, limit, slowdown, or Z-index inputs to determine the zero-reference position (home point) used for absolute positioning. The Commander core has thirteen (13) different built-in home search routines. Since the signal or signals used to define the home point varies depending on the homing mode selected, carefully select a homing mode that best matches the environment and available input signals of your system.

Some homing routines that involve a decelerated stop may result in a final position that is non-zero. The zero-reference position is correctly preserved as the position that was marked when the home trigger condition was detected.

In ASCII mode, use the **H[axis][+/-][mode]** command to perform a home move. Standalone mode will use the **HOME[axis][+/-][mode]** command to perform a homing routine. Please refer to Appendix A: Homing Mode Actions for a full list of the available homing routines and a diagram of the inputs required and their timing.

When **ERC[axis](bit1)** is set to 1, the **ERC**(deflection counter clear) signal is output when the zero-reference position (home point) is reached. For more information on the ERC timing see homing operation in **Appendix A: Homing Mode Actions**.

ASCII	H[axis][+/-][mode]	ERC[axis]
Standalone	HOME[axis][+/-][mode]	—

2.2.5.1 Z-move Operation

The Commander core has a built-in move called a Z-Move. When the **ZMOVE[axis][dir]** command is sent the motor will move in the direction commanded at **HSPD** until such time as the number of Z-index pulses are detected as set by the **ZCNT[axis]=[0-15]** command.

The **ZCNT[axis]** command also works with any homing operation that uses the Z-index pulses as part of the homing operation to set the number of Z-index pulses that need to be detected. The range for **ZCNT[axis]** is 0-15. The number of Z-index pulses to be detected will be the number set plus one. **Example:** 0 = stop at the 1st Z-index detection, 1 = stop at the 2nd Z-index detection, 15 = stop at the 16th Z-index detection.

ASCII	ZMOVE[axis][dir]	ZCNT[axis]
Standalone	ZMOVE[axis][dir]	—

Operating procedure

If there is no change in the previous set data, the operations 1 and 2 below are not necessary.

1) Set the operating speeds
2) Set the Z-index count numbers and the Z-index input logic
3) Issue a homing command

Operating conditions

- Joystick motion is disabled
- MPG operation is disabled
- Operation is stopped
- The error status is clear

Related commands

LSPD, HSPD, ACC, DEC, H, ZCNT, MST

2.3 Interpolation Operation

Interpolated operations involve two or more axes whose movements are interdependent. There are many types of interpolated moves. **Linear interpolation** is when two or more axes works together to cause movement along a straight line. For linear interpolation, each axis moves in only one direction, and start and stop at the same time. **Circular interpolation** is when two axes work together to cause movement along the outer rim of a circle. For circular Interpolation, each axis will change direction and speed as required to allow for a circular path of movement. A popular subset of circular interpolation is **arc interpolation,** when two axes work together to move along the outer rim of a circle for a set number of degrees. **Helix or tangential interpolation** is when a circular or arc interpolation of the X and Y-axis is combined with linear interpolation of the center point of the circle and the Z-axis. This allows for movement along a helical motion path. The Commander core allows for all of these different moves, independently or in combination. These moves can be **buffered** together to allow for seamless **continuous** motion.

For more details on how the Commander core handles interpolation, refer to **Appendix C - Interpolation**.

2.3.1 Linear interpolation

The Commander core can perform linear interpolated motion for up to 4 axes. The **X**, **Y**, **Z**, and **U** commands can be combined into a single move command that consists of up to 4 target positions (one for each axis). Any combination of axes can be used for linear interpolation, and at least two axes must be referenced in order to perform linear interpolation.

By default, interpolation is enabled (**EINT=1**). While interpolation is enabled, linear interpolation operations are performed by writing positioning commands of multiple axes on the same line. There is no dedicated command for linear interpolation. If Interpolation is disabled (**EINT=0**), writing positioning commands of multiple axes on the same line will not result in an interpolated move. See Figure 2-20.

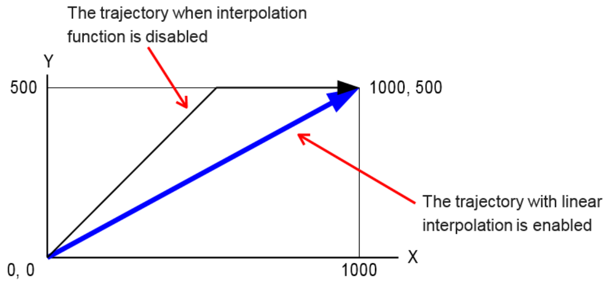


Figure 2-20

For example, the **X1000Y1000Z100U800** will move all four axes to the position (1000, 1000, 100, 800).
Similarly, the **X1000Y-1000Z500** will move the X, Y, and Z-axis to the position (1000, -1000, 500) while the U axis remains idle.

The Commander core will output pulses to move the motors immediately upon receipt of the command. While performing a linear interpolated motion, any axis not included in the linear interpolated motion can be moved independently at the same time, or included in other interpolated motions, without impacting the linear interpolated motion.

For more details on how the Commander core handles interpolation refer to **Appendix C - Interpolation**.

ASCII	X[target]	Y[target]	Z[target]	U[target]	EINT
Standalone	X[target]	Y[target]	Z[target]	U[target]	EINT

Operating procedure

If there is no change in the previous set data, operations 1 to 3 below are not necessary

1) Set an operating speed
2) Set the positioning coordinate mode with **ABS** or **INC** command
3) Enable an interpolation operation with the **EINT** command
4) Issue a positioning operation command of multiple axes

Operating conditions

- Joystick motion is disabled
- MPG operation is disabled
- Operation is stopped
- The error status is clear

Related commands

LSPD, HSPD, ACC, DEC, EINT, X, Y, Z, U, MST

2.3.2 Circular interpolation

Circular interpolation is when two axes work together to cause movement along the outer rim of a circle. The Commander core supports circular interpolation moves using the **CIRP** and **CIRN** commands. This will perform a complete 360° circle using any two axes. When less than 360° of motion is needed, then arc interpolation is used with the **ARCP** and **ARCN** commands. The two functions can be combined when more than 360° of motion is needed. Circles and arcs can be drawn using any two axes.

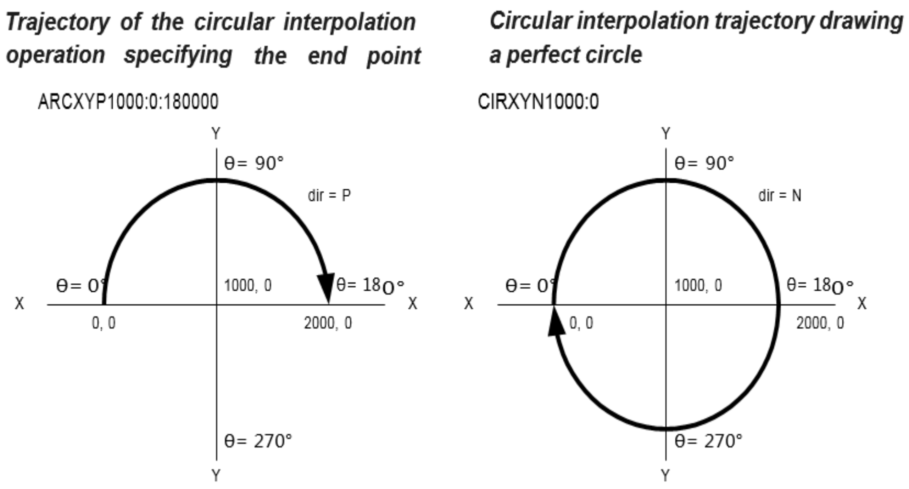


Figure 2-21

CIR[A1][A2]P[C1]:[C2] – Draw a circle in CW direction where [A1][A2] signifies the selected axes and [C1][C2] signifies the absolute position of the circle center.

CIR[A1][A2]N[C1]:[C2] – Draw a circle in CCW direction where [A1][A2] signifies the selected axes and [C1][C2] signifies the absolute position of the circle center.

For example, the command **CIRXZP1000:1000** will move the X and Z axis in a full circle in the CW direction with the XZ position (1000, 1000) as the circle center.

Circular interpolated moves will use the current axis positions as well as the specified circle center to automatically calculate the circle radius. The maximum allowable radius is 134,216,773 pulses for circular interpolated moves. All circular moves are interpreted as absolute moves.

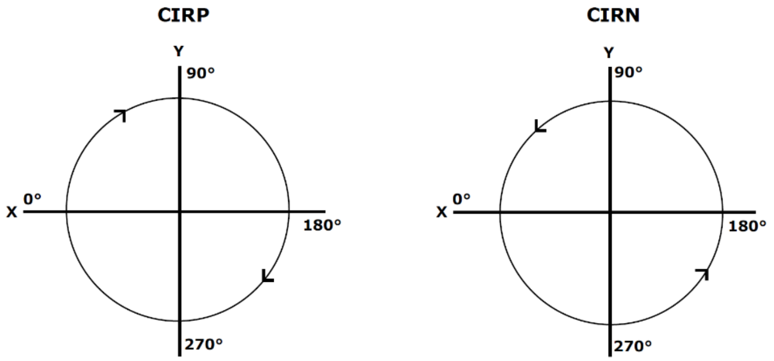


Figure 2-22

2.3.2.1 Arc interpolation

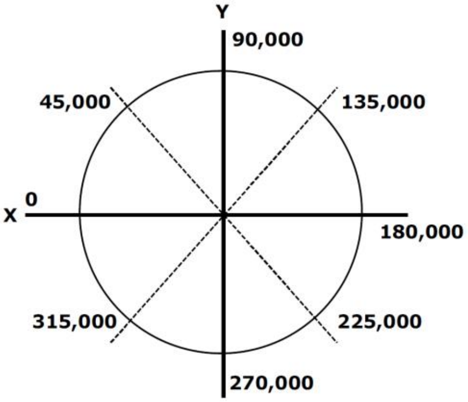


Figure 2-23

The Commander core supports arc interpolation moves using the **ARCP** and **ARCN** commands.

ARC[A1][A2]P[C1]:[C2]:[θ] – Draw an arc in CW direction where[A1][A2] signifies the selected axes, [C1][C2]signifies the absolute position of the arc center, and θ signifies the absolute arc angle.

ARC[A1][A2]N[C1]:[C2]:[θ] – Draw an arc in CCW direction where [A1][A2] signifies the selected axes, [C1][C2]signifies the absolute position of the arc center, and θ signifies the absolute arc angle.

An arc interpolated move will use the current axis positions as well as the specified arc center to automatically calculate the radius. The maximum allowable radius is 134,216,773 pulses on arc interpolated moves. All arc moves are interpreted as absolute moves.

The absolute angle (θ) is in units of a milli-degree. For example, to move to the absolute angle of 45°, the absolute angle parameter would be 45000.The absolute angle locations can be found in Figure 2-23.

Figure 2-24 shows the difference between issuing the **ARCXYP0:0:90000** and the **ARCXYN0:0:90000** command if both cases have the same starting point at 0 degrees.

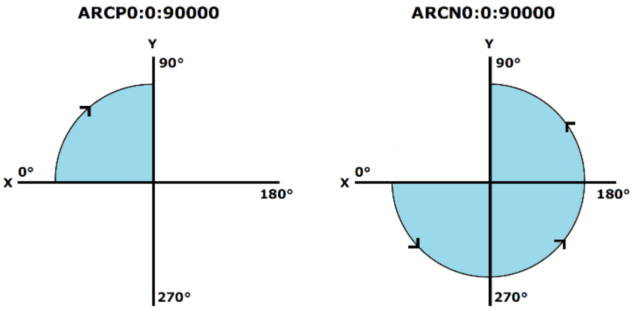


Figure 2-24

For more details on how the Commander core handles interpolation refer to Appendix C - Interpolation.

ASCII	CIR[A1][A2]P[C1]:[C2]	CIR[A1][A2]N[C1]:[C2]
Standalone	CIR[A1][A2]P[C1]:[C2]	CIR[A1][A2]N[C1]:[C2]

ASCII	ARC[A1][A2]P[C1]:[C2]:[θ]	ARC[A1][A2]N[C1]:[C2]:[θ]
Standalone	ARC[A1][A2]P[C1]:[C2]:[θ]	ARC[A1][A2]N[C1]:[C2]:[θ]

Operating procedure

If there is no change in the previous set data, operations 1 to 3 below are not necessary.

1) Set an operating speed
2) Set the positioning coordinate mode with **ABS** or **INC** command
3) Enable the interpolation operation with the **EINT** command
4) Issue a circular interpolation operation command

Operating conditions

- Joystick motion is disabled
- MPG operation is disabled
- The operation is stopped
- The error status is clear

Related commands

LSPD, HSPD, ACC, DEC, EINT, ARC, CIR, MST

2.3.3 Helical or Tangential interpolation

The Commander core is able to combine the linear and circular interpolation operations in order to perform helix or tangential type moves. These moves can only use the X and Y axes for the circular interpolation, with the Z axis used for linear coordination.

A move with less than 360° movement for the X and Y axis is made by using the **ARCTP** and **ARCTN** commands. A move with 360° movement for the X and Y axis is made by using the **CIRTP** and **CIRTN** commands. A continuous movement with more than 360° of movement is possible by using the buffer operation to combine the **ARCTP/ARCTN** and **CIRTP**/**CIRTN** commands.

ARCTP[C1]:[C2]:[θ]:[Z target] – Draw an arc in CW direction where [C1][C2] signifies the absolute position of the arc center, θ signifies the absolute arc angle, and [Z target] signifies the absolute end point.

ARCTN[C1]:[C2]:[θ]:[Z target] – Draw an arc in CCW direction where [C1][C2] signifies the absolute position of the arc center, θ signifies the absolute arc angle, and [Z target] signifies the absolute end point.

CIRTP[C1]:[C2]:[Z target] – Draw a circle in CW direction where [C1][C2] signifies the absolute position of the circle center and [Z target] signifies the absolute end point.

CIRTN[C1]:[C2]:[Z target] – Draw a circle in CCW direction where [C1][C2] signifies the absolute position of the circle center and [Z target] signifies the absolute end point.

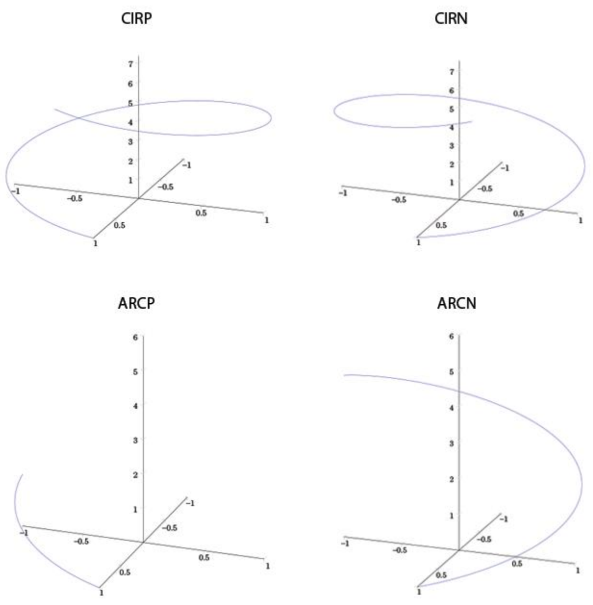


Figure 2-25

Note that these commands will use the U axis for calculation purposes; therefore, the U axis should NOT be used for motion while using these commands.

For more details on how the Commander core handles interpolation refer to **Appendix C - Interpolation**.

ASCII	ARCTP[C1]:[C2]:[θ]:[Z target]	ARCTN[C1]:[C2]:[θ]:[Z target]
Standalone	ARCTP[C1]:[C2]:[θ]:[Z target]	ARCTN[C1]:[C2]:[θ]:[Z target]

ASCII	CIRTP[C1]:[C2]:[Z target]	CIRTN[C1]:[C2]:[Z target]
Standalone	CIRTP[C1]:[C2]:[Z target]	CIRTN[C1]:[C2]:[Z target]

2.3.4 Buffered interpolation operation

The Commander core's buffer mode allows for up to 100 prebuffered interpolated moves. This allows for the creation of unique patterns without any delay between moves. Each move has its own constant speed setting.

For normal operation, commands are written to an operation register.

For buffer operation, the coordinated motion commands are written to the buffer registers instead. There are 100 separate buffer registers to hold coordinated motion commands. When the start buffer motion (**BSTART**) command is sent, the coordinated motion command in Buffer0 is moved to the operation register. Then the coordinated motion command in Buffer1 is moved to Buffer0 and so on. This process is continued until all the buffer registers are empty or the buffer mode is turned off with the (**BF**) command. See table below for details.

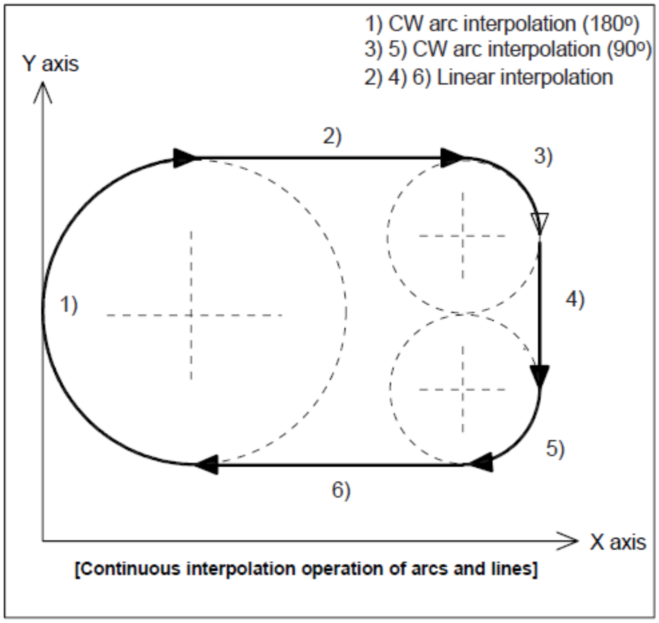

1) CW arc interpolation (180º)
3) 5) CW arc interpolation (90º)
2) 4) 6) Linear interpolation
Y axis
1)
2)
3)
4)
5)
6)
X axis
[Continuous interpolation operation of arcs and lines]

Figure 2-26

Operation register		For normal operation, a command is written in the operation register, and the operation starts immediately.
Buffer0	100 Buffer registers	For buffer operations, coordinated motion commands are written to the buffer registers. When buffer operation starts, these commands are automatically written to the operation register, which automatically starts the operation, and be deleted from the buffer registers. Buffered commands are processed on a first-in first-out basis.
Buffer1		
Buffer2		
⋮		
Buffer97		
Buffer98		
Buffer99		
New Commands		As the buffered commands are moved to the operating register, the buffer registers can be loaded with additional coordinated motion commands.

To turn the buffer mode on and off, use the **BO** and **BF** ASCII commands, respectively. For standalone mode, the **BUFON** and **BUFOFF** commands should be used. The buffered move operation cannot be used while StepNLoop is enabled.

When buffer mode is enabled, motion will not start until the appropriate command has been sent, even if there are commands in the buffer. This allows the buffer to be fully loaded before any move is processed. The command **BSTART** starts feeding the buffered commands to the operation register.

The status of the buffer operation can be read using the **BSTAT** command. The return value will contain the buffer enable status, buffer start and end positions, and the number of available buffer registers. The return value has the following format:

[Buffer enabled]:[Buffer start]:[Buffer end]:[Available Buffer]

The buffer operation only supports motion of the X, Y, and Z-axis. In addition, only linear, circular and arc interpolation moves can be added to the buffer. As such only the linear, circular and arc interpolation commands below are supported during buffered operation.

Linear interpolation

In ASCII mode, a buffered move command will have the syntax **I[X pos]:[Y pos]:[Z pos]:[speed]**. For example, the **I1000:2000:3000:5000** command indicates position (1000, 2000, 3000) at a speed of 5000. If buffered mode is not enabled, the **I** command will not be processed by the controller.

In standalone mode, the buffered move command will have the syntax **X[pos]Y[pos]Z[pos]**, and the speed for the move command will use the current global high-speed setting. If buffered mode is not enabled, these standalone commands will be processed as linear interpolation commands.

Circular and Arc interpolation

The **ARC** and **CIR** commands, described in Section 2.3.2 Circular interpolation, can be used when buffer mode is enabled. The syntax for these commands is the same in ASCII and standalone mode.

ASCII	BO	BF	BSTART	BSTAT	I[X]:[Y]:[Z]:[S]
Standalone	BUFON	BUFOFF	ISTART	—	X[pos]Y[pos]Z[pos]

Operating procedure

1) Enable buffer operation with **BO** commands.
2) **I** command writes an interpolation operation (**CIR** or **ARC** command is also available)
3) Buffer operation is started with a **BSTART** command
4) Add the interpolation operation while checking the buffer register free space with the **BSTAT** command
5) If there is no interpolation operation to be added, the buffer operation needs to be disabled by the **BF** command after completion

Operating conditions

- Joystick motion is disabled
- MPG operation is disabled
- Corrective action is disabled
- The error status is clear

Related commands

BO, BF, BSTAT, BSTART, I, CIR, ARC

2.4 On-the-fly-speed/target position change

The Commander core supports the ability to change the moving speed and/or the target position on the fly.

2.4.1 On-the-fly speed change

This function can change the speed while an axis is in motion, at any point in the motion profile. Figure 2-27, shows a few examples of how speed changes are executed.

If the target speed is changed during the acceleration process, and:

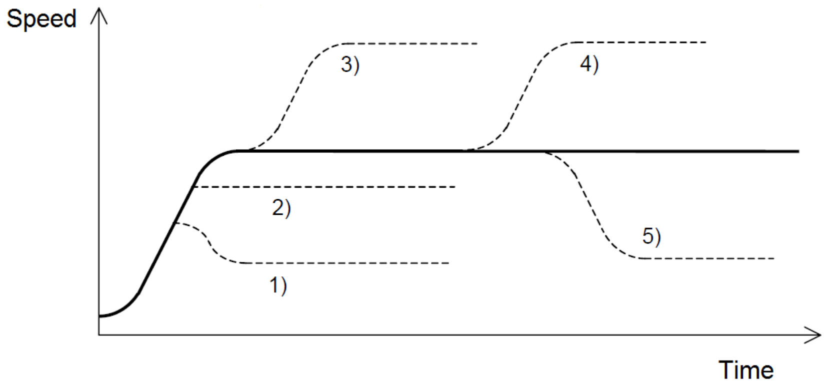


Figure 2-27

1) If the newly set value is lower than the pulse rate at the time of the change, deceleration is made to the newly set value.
2) If the newly set value is equal to or higher than the pulse rate at the time of the change but less than the previous target, acceleration is made to the newly set value.
3) If the newly set value is greater than the previous target, acceleration is made to the preset pulse rate and then to the newly set value.

If the target speed is changed while traveling at the previously set target speed:

4) If the new value is higher than the previous target, acceleration is made to the newly set value.
5) If the new value is lower than the previous target, deceleration is made to the newly set value.

On-the-fly speed change can be achieved using the **SSPD[axis]=[newspeed]** command. Before executing an on-the-fly speed change, you need to ensure the correct speed range is set with the **SSPDM[axis]** command. For more information on the speed range setting see **Appendix B: Speed Settings** and SSPDM in the command refence.

ASCII	SSPD[axis]=[newspeed]	SSPDM[axis]
Standalone	SSPD[axis]=[newspeed]	SSPDM[axis]

Operating procedure

If there is no change in the previous set data, operation 1 below is not necessary.

1) Set the speed range with SSPDM command
2) Issue the speed change command

Operating conditions

- While operating at HSPD (not decelerating)
- Buffer operation must disable
- S-curve acceleration/deceleration mode must disable

Related commands

SSPD, SSPDM, MST, PS

2.4.2 On-the-fly target position change (Target position change)

On-the-fly target position change can be achieved using the **T[axis][newtarget]** command, and will change the final destination of the motor while the motor is moving. If the motor has already passed the new target position, it will reverse direction when the target position change command is issued.

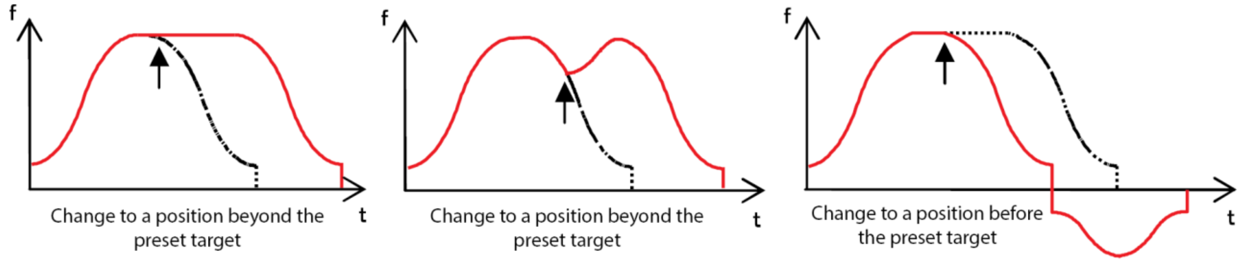


Figure 2-28

An on-the-fly target position change command will only be valid if the specified axis is performing an individual position move. An interpolated, jogging, or homing move will not accept an on-the-fly target position change command.

In **ABS** mode, the target change will be to the absolute position from the homed zero point. In **INC** mode the target change will be to the incremental position from the original move start point.

ASCII	T[axis][newtarget]
Standalone	—

Operating procedure

There is no predetermined procedure. On-the-fly target position change commands can be issued while the axis is moving

Operating conditions

- While operating with the set value of HSPD (not decelerating)

Related communication commands

T, MST

2.5 Advanced Functions

2.5.1 Joystick operation

Joystick control is available on the Commander core for the X and Y-axis. When this mode is enabled, the pulse speed and direction output for the X and Y axis can be controlled by their corresponding analog input. See the axis to analog input assignment in Table 2-23 below.

Axis	Analog Input
X	AI1
Y	AI2

Table 2-23 Analog inputs

To enable or disable joystick control for an axis, use the ASCII command **JENA** or the standalone command **JOYENA**. The joystick enable parameter is a 2-bit value. For example, A joystick enable value of 3 means the joystick feature is enabled on both the X and Y-axis. If joystick control is enabled, StepNLoop is automatically disabled.

Bit	Description	Setting	
0	Joystick control X-axis	0 - Disabled	1 - Enabled
1	Joystick control Y-axis	0 - Disabled	1 - Enabled

Table 2-24 JENA command

For the Commander core to properly translate the analog inputs into a safe pulse speed and direction output for your system, you first must define what is acceptable and safe for your system. This includes defining the relation between voltage and **direction** of travel, and the zero-tolerance (or dead) zone during which there is no output from the Commander core. The **safe speeds** are defined as the maximum speed the analog input can command and maximum rate of speed change. The joystick **positioning** soft limits define where the analog inputs can command a system to move.

2.5.1.1 Direction control

The **JMAX[axis]** command can be used to define the maximum analog input value for the specified axis. This parameter has units of millivolts and a range of 0 to 3300. This value will define the analog input value in which the maximum joystick speed occurs in the positive direction. This command is only available in ASCII mode.

Similarly, the **JMIN[axis]** command can be used to define the minimum analog input value for the specified axis. This parameter has units of millivolts and a range of 0 to 3300. This value will define the analog input value in which the maximum joystick speed occurs in the negative direction. This command is only available in ASCII mode.

MID represents the zero-joystick position. There is no command corresponding to *MID*. This is an internally calculated value using the following formula:

$$MID = ((\mathbf{JMAX}[\mathbf{axis}] - \mathbf{JMIN}[\mathbf{axis}]) / 2) + \mathbf{JMIN}[\mathbf{axis}]$$

During joystick operation, analog input of **JMIN[axis]** to *MID* represents negative joystick direction and analog input of *MID* to **JMAX[axis]** represents positive joystick direction. Setting the value of **JMAX[axis]** to be less than the value of **JMIN[axis]** will result in a change of direction polarity during joystick operation.

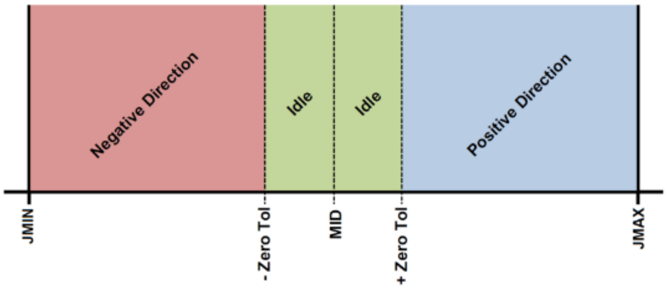


Figure 2-29

The ASCII command **JTOL[axis]** can be used to define the zero-tolerance (or dead) zone around the *MID* value. No movement will occur while the analog input value is within the zero-tolerance zone. This parameter has units of millivolts and a range of 0 to 3300. The zero-tolerance parameter will be on the positive and negative side of the *MID* point to define to the zero-tolerance zone. See Figure 2-29 for details.

Example: The zero-tolerance zone at **JMAX** = 3,250 mV, **JMIN** = 10 mV, **JTOL** = 50 mV will be

- 1580 mV ~ 1680 mV since ((3,250 – 10) ÷ 2) +10 = 1,630 mV
- 1,630 mV ±50 mV

2.5.1.2 Speed control

The maximum joystick speed for the X and Y-axes is set using the ASCII command **JSPD[axis]**. For standalone mode, the **JOYHS[axis]** command should be used. This parameter will define the speed of the specified axis at the maximum and minimum analog input values.

The maximum allowable speed change (delta) for the X and Y axes is set using the ASCII command **JDEL[axis]**. For standalone mode, the **JOYDEL[axis]** command is used. This parameter will control the acceleration/deceleration of the specified axis while the joystick mode is enabled.

2.5.1.3 Position control

Joystick control also has soft limits, which are defined by a negative outer limit, negative inner limit, positive inner limit, and a positive outer limit, using the **JLIM[number]** command. The soft limits are in units of pulses.

When moving in a positive direction, as soon as the positive inner limit is crossed, the speed is reduced. Speed will continue to decline as the position moves closer to the outer limit value. If the position reaches the positive outer limit, the joystick speed is set to zero and movement in the positive direction is prohibited. The same behavior is applicable to the negative direction and negative limits.

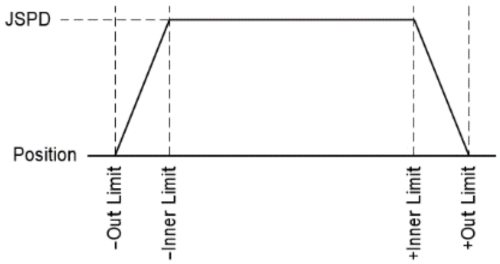


Figure 2-30

Figure 2-30 represents the relationship between the joystick speed and inner and outer limits.

Table 2-25 JLIM command below shows the command assignment of the negative and positive soft limits for joystick control, as well as a summary of the additional joystick commands.

Command	Description
JLIM1	X-axis negative outer limit
JLIM2	X-axis negative inner limit
JLIM3	X-axis positive inner limit
JLIM4	X-axis positive outer limit
JLIM5	Y-axis negative outer limit
JLIM6	Y-axis negative inner limit
JLIM7	Y-axis positive inner limit
JLIM8	Y-axis positive outer limit

Table 2-25 JLIM command

This limit value is based on the value "0" of the command pulse counter. If a stop command is issued while a joystick operation is active, it will return an invalid status.

ASCII	JENA	JMAX[axis]]	JMIN[axis]	JSPD[axis]	JDEL[axis]	JLIM[number]
Standalone	JOYENA	—	—	JOYHS[axis]	JOYDEL[axis]	—

Operating procedure

If there is no change in the previous set data, operation 1 below is not necessary.

1) Set all the joystick operation values
2) Enable joystick operation with JENA command
3) Move the axis by operating the joystick
4) Disable the joystick operation at the end of the operation

Operating conditions

- Joystick operation is disabled
- MPG operation is disabled
- Operation is stopped
- The error status is clear

Related communications commands

JSPD, JDEL, JMAX, JMIN, JTOL, JENA, JLIM, AI, MST

2.5.2 Manual pulse generator (MPG) operation

The MPG input can be used for a number of functions such as:

1) A manual pulse generator, which allows for fine adjustment of an axis.
2) Cam follower, which will allow the axis to follow a cam axis based on its encoder output.
3) Electronic gearing, which allows an axis to follow the input commands but at a different rate of speed (faster or slower).

The Commander core has a dedicated MPG input for each axis. To enable the MPG input, set the **MPE[axis]** command to 1. To disable the MPG input, set the **MPE[axis]** command to 0.

The MPG inputs are controlled by three commands: MPG signal input mode (**POL[axis]**), multiplication (**MPM[axis]**) and division (**MPD[axis]**). These work together to set how the pulses from the MPG input controls the motion of an axis.

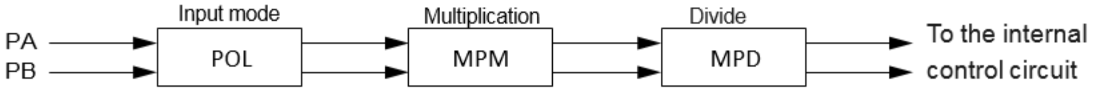


Figure 2-31

2.5.2.1 MPG signal input mode

MPG signal input mode

The Commander core can accept two different pulse types which can be set using the **POL[axis](Bit15~16)**.

1) Two pulse type (not common) is where pulses are sent to A input to count up and B input to count down. Each pulse input is one count of the encoder.

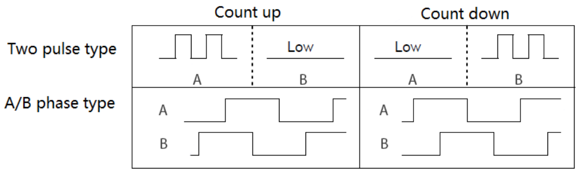


Figure 2-32

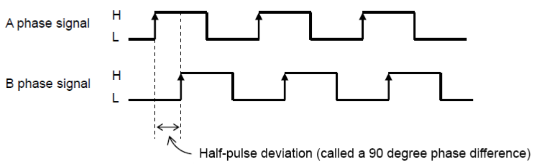


Figure 2-35

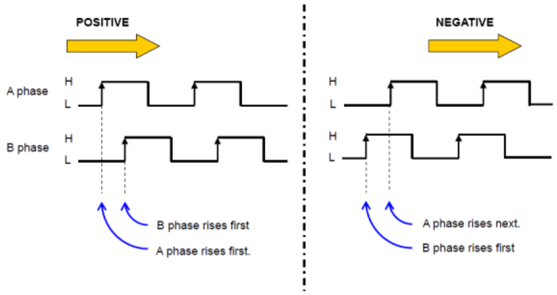


Figure 2-35

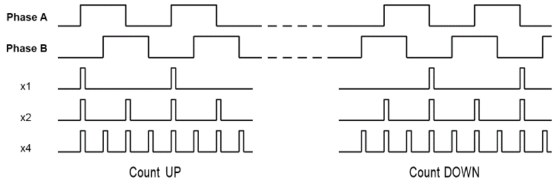


Figure 2-35

2) A/B phase type (common).

   a. For an A/B phase type there should be 90° difference between the pulses from A and B. See Figure 2-35.

   b. By default, when the A phase leads the B phase motion is in the positive direction, when B phase leads the A phase motion is in the negative direction. The direction can be changed by **POL[axis](Bit14)**. See Figure 2-35.

   c. For an A/B phase type you can select x1, x2 or x4 multiplication rate. Figure 2-35 shows how the A/B phase counts would be interpolated for a x1, x2, and x4 multiplication rate.

      i. x1 multiplication – Only the rising edge of the A phase is counted, B phase is not counted and only used to check the direction of motion.

      ii. x2 multiplication – The rising and falling edge of the A phase is counted, B phase is not counted and only used to check the direction of motion.

      iii. x4 multiplication – The rising and falling edge of the A and B phase is counted, and used to check the direction of motion.

Maximum frequency input on the MPG line is the maximum pulses that can be input to the MPG input, not the counts. Since most manual pulse generator outputs are rated based on a x4 multiplication rate, the output frequency of pulses is most likely ¼ the count rate output. Please check the datasheet for your manual pulse generator to confirm.

2.5.2.2 Electronic gearing the MPG signal

After the input mode stage, the MPG pulse signal is converted to counts based on the **POL[axis]** settings. The counts then pass through an electronic gearing stage, which includes a multiplier **MPM[axis]** and divider **MPD[axis]** circuit.

MPM[axis] can be used to access the multiplier. The multiplier can be set to a value from [1 to 32]. A value of 1 will turn off the multiplier.

MPD[axis] is the numerator of the divider circuit. The **MPD[axis]** can be set to a value from [1 to 2048]. The denominator of the divider circuit is fixed as 2048. A value of 2048 will turn the divider off.

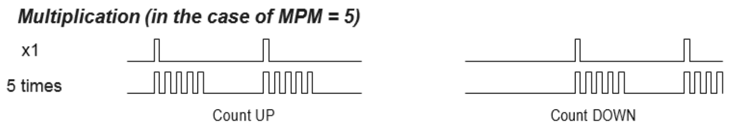


Figure 2-36

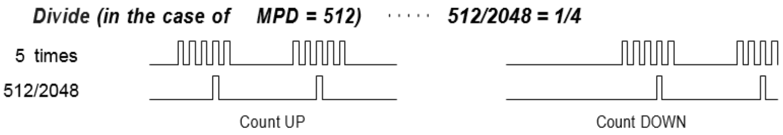


Figure 2-37

The multiplier and divider circuits can be used together to create any ratio. The multiplier and divider use the following equation to determine the number of output counts[3]:

$$\text{Output counts} = (\text{MPG counts}) * \frac{\text{MPM} * \text{MPD}}{2048}$$

Example: Final ratio that is needed is 100 pulses-in and 125 pulses-out (1.25). By setting MPM[axis] to 5, then the input of 100 pulses is 500. (see Figure 2-36) since 500/4 = 125 MPD[axis] = 512 (512/2048 = ¼) the output pulses will be 125 (see Figure 2-37).

$$125 = (100) * \frac{5 * 512}{2048}$$

MP[axis] can be written to set an MPG position or read to see the current position. The counter value here is the final counts after the electronic gearing stage.

ASCII	MPD[axis]	MPE[axis]	MPM[axis]	MP[axis]
Standalone	—	MPGE[axis]	—	—

Operating procedure

If there is no change in the previous set data, operation 1 below is not necessary.

1) Set the signal input mode and the magnification, etc.
2) Enable the MPG operation with MPE command
3) Input the MPG signal and operate
4) Disable the MPG function at the end of operation.

Operating conditions

- Joystick motion is invalid
- Operation is stopped
- The error status is clear

Related commands

MPE, MPD, MPM, MP, MST

[3] MPG counts are counts after the input mode setup done by POL[axis].

2.5.3 StepNLoop Closed Loop Control operation

The Commander core features a closed-loop position verification algorithm called StepNLoop. The algorithm requires the use of an incremental encoder.

StepNLoop performs the following operations:

1) **Position Verification:** At the end of any targeted move, StepNLoop will perform a correction if the current error is greater than the tolerance value (**SLT[axis]**).
2) **Delta Monitoring:** The delta value is the difference between the actual and the target position. When delta exceeds the error range value (**SLE[axis]**), the motor is stopped and the StepNLoop status goes into an error state[4]. Delta monitoring is performed during moves – including homing and jogging. To read the delta value, use the **D[axis]** command.

StepNLoop is commonly used to close a position loop on stepper systems (encoder on back of motor). StepNLoop can also be used for the external position loop on stepper or servo systems, which have a loss of motion due to backlash or hysteresis, as long as the encoder feedback to the Commander core is from the final work point instead of the motor. See Figure 2-39 Encoder at motor and Figure 2-39 Encoder at work point, as examples.

This table outlines how StepNLoop behaves in different scenarios.

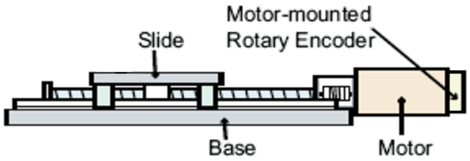


Figure 2-39 Encoder at motor

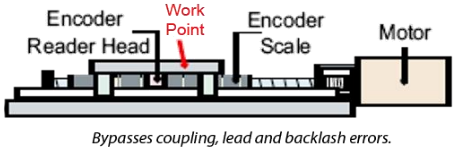


Figure 2-39 Encoder at work point

Condition	StepNLoop behavior (motor is moving)	StepNLoop behavior (motor is idle)
δ ≤ SLT	Continue to monitor the **D[axis]**	In Position. No correction is performed.
δ >SLT AND δ < SLE	Continue to monitor the **D[axis]**	Out of Position. A correction is performed.
δ >SLT AND δ > SLE	Stall Error. Motor stops and signals an error[4].	Error Range Error. Motor stops and signals an error[4].
Correction Attempt > SLA	N/A	Max Attempt Error. Motor stops and signals an error[5].

Table 2-26 StepNLoop behavior

Key:

[δ]:	Error between the target position and actual position
SLT:	Tolerance range
SLE:	Error range
SLA:	Max. correction attempt

[4] The error status here is not the main status (**MST**) but the status (**SLS**) of the compensation operation.

[5] The error status here is not the main status (**MST**) but the status (**SLS**) of the compensation operation.

In order to perform the StepNLoop Closed Loop Control operation, the following four parameters must be set beforehand. Parameters should be set for each axis.

StepNLoop Parameter	Description	Command
StepNLoop Ratio	The ratio between motor pulses and encoder counts. This ratio will depend on the motor type, micro-stepping, encoder resolution, and decoding multiplier. The value must be in the range [0.001, 999.999].	SLR[axis]
Tolerance	The maximum error between target and actual position that is considered "In Position." In this case, no correction is performed. Units are in encoder counts.	SLT[axis]
Error Range	The maximum error between target and actual position that is not considered a serious error. If the error exceeds this value, the motor will stop immediately and go into an error state.	SLE[axis]
Correction Attempt	The maximum number of correction tries that the controller will attempt before stopping and going into an error state.	SLA[axis]

A convenient way to find the StepNLoop ratio is to set EX=0, PX=0 and move the motor +1000 pulses. The ratio can be calculated by dividing 1000 by the resulting EX value. Note that the value must be positive. If it is not, then the direction polarity must be adjusted. This test should be performed while StepNLoop is disabled.

To enable/disable the StepNLoop feature use the **SL[axis]** command. To read the StepNLoop status, use **SLS[axis]** command. See Table 27 SL command below for a list of the return values.

Value	Description
0	Idle
1	Moving
2	Correcting
3	Stopping
4	Aborting
5	Jogging
6	Homing
7	Z-Homing
8	Correction range error. To clear this error, use **CLR** command.
9	Correction attempt error. To clear this error, use **CLR** command.
10	Stall Error. **D** value has exceeded the correction range value. To clear this error, use **CLR** command.
11	Limit Error
12	N/A (i.e. StepNLoop is not enabled)
13	Limit homing

Table 27 SL command

While StepNLoop is enabled, position move commands are in term of encoder position. For example, **X**1000 means to move the motor to encoder position 1000. This applies to individual as well as interpolated moves.

Additionally, once StepNLoop is enabled, the speed is in encoder speed. For example, **HSPD=**1000 when StepNLoop is enabled means that the target high speed is 1000 encoder counts per second. This only applies to individual axis moves.

For linear interpolated movement, the speed is calculated as pulse/sec, NOT encoder counts/sec. The same will apply to the X and Y axis while performing arc/circular interpolated moves.

ASCII	SL[axis]	SLR[axis]	SLT[axis]	SLE[axis]	SLA[axis]	SLS[axis}
Standalone	SL[axis]	—	—	—	—	SLS[axis}

ASCII	D[axis]	CLR[axis]
Standalone	—	—

Operating procedure

If there is no change in previous set data, operation 1 below is not necessary.

1) Set each setting value of StepNLoop Closed Loop Control operation.
2) Enable the StepNLoop Closed Loop Control operation function with the **SL** command.
3) Execute a positioning operation.
4) A StepNLoop Closed Loop Control operation is executed after the positioning operation is completed.

Operating conditions

- Joystick operation is invalid
- Buffer operation is invalid
- The error status is clear

Related commands

SL, SLR, SLT, SLE, SLA, SLS

2.6 Status monitoring function

The Commander core has a number of registers that can be queried at any time for the current system status.

2.6.1 Counter values

The Commander core has three counters for each axis that can be read to obtain the current number of output pulses, encoder pulses received, and the deviation between the output and input.

2.6.1.1 Output Pulse Counter

This counter shows the total number of pulses that have been to the motor driver. It can be read using the ASCII command **PP**, which returns the pulse count of all 4 axes. The return value has the following format:

[Pulse X]:[Pulse Y]:[Pulse Z]:[Pulse U]

The pulse count of each axis can also be accessed individually. To manually set/get the pulse position of an individual axis, use the **P[axis]** command.

When StepNLoop closed-loop control is enabled, the pulse counter commands return the real-time target position of the motor. When StepNLoop closed-loop control is disabled the pulse position commands return the step position. See 2.5.3 StepNLoop Closed Loop Control operation for details on the StepNLoop feature.

2.6.1.2 Feedback Pulse Counter

This counter shows the total number of pulses that have been received on the encoder input. It can be read using ASCII command **EP**, which returns the encoder counts of all 4 axes. The return value has the following format:

[Encoder X]:[Encoder Y]:[Encoder Z]:[Encoder U]

The encoder count of each axis can also be accessed or set individually, by using the **E[axis]** command.

2.6.1.3 Deviation Counter

This counter is enabled with StepNLoop operations, and shows the difference between commanded position and the encoder input. To get the deviation counts of an individual axis, use the **D[axis]** command. When the StepNLoop function is disabled, the value of **D[axis]** will be 0.

ASCII	P[axis]	PP	E[axis]	EP	D[axis]
Standalone	P[axis]	—	E[axis]	—	—

Operating procedure

There is no particular condition.

Related commands

P, PP, E, EP, D

2.6.2 Axis status register

The Commander core has two axis-monitoring registers for each axis; these can be read to obtain the current status of each axis.

2.6.2.1 Axis Status Acquisition Command

Motor/axis status register can be read anytime using **MST[axis]** command. The reply contains information about the motor status, including error status such as end limit stop, alarm stop, and EMG stop. Value of the motor status is relayed as an integer value with the following bit assignment:

Bit	Function	Condition	
0	Accelerating	0: Not Accelerating	1: Accelerating
1	Decelerating	0: Not Decelerating	1: Decelerating
2	Constant Speed	0: Not at Constant Speed	1: At Constant Speed
3	Alarm Signal Input Status	0: OFF	1: ON
4	Positive End Limit Status	0: OFF	1: ON
5	Negative End Limit Status	0: OFF	1: ON
6	Home or Origin Status	0: OFF	1: ON
7	Slowdown Input Status	0: OFF	1: ON
8	Positive End Limit Error[6]	0: No Error	1: Error
9	Negative End Limit Error[6]	0: No Error	1: Error
10	Alarm Error[6]	0: No Error	1: Error
11	In-Position Input Status	0: OFF	1: ON
12	Deviation Counter Clear	0: OFF	1: ON
13	Z-index Input Status	0: OFF	1: ON
14	External Status Input	0: OFF	1: ON
15	EMG Signal Status	0: OFF	1: ON
16	EMG Error Status[6]	0: No Error	1: Error
17	Stop by Slowdown Detection[7]	0: Not Stopped	1: Stopped
18	Wait for In-Position Input Status	0: Not Waiting	1: Wait
19	Wait for External Start Signal Input	0: Not Waiting	1: Wait

Table 2-28 MST command

Additional information on this register can be found in the command reference.

2.6.2.2 Pulse Speed register

The current pulse rate register can be read using the command **PS[axis]**. For units, see the table below. Section 2.5.3 StepNLoop Closed Loop Control operation contains more information.

Operation Mode	Speed Units
StepNLoop disabled	Pulse / sec
ALL interpolated moves	Pulse / sec
StepNLoop enabled and non-interpolated move	Encoder counts / sec

Table 2-29 PS command units

[6] For Bits 8, 9, 10, and 16 (+ end limit, - end limit, alarm, and EMG errors), errors must be cancelled with the **CLR** command before next operation.

[7] For Bit 17 (Slowdown Detection), additional operations can be performed with flag ON (1) but the flag will remain ON until canceled by **CLR** command.

2.6.2.3 Error clear command

Many errors in the Commander core will set flags to alert to the fact that an error occurred. These error flags can be cleared with use of the **CLR** ASCII command. For standalone mode, the **ECLEAR[axis]** command should be used.

This command clears the error flags one axis at a time; there is no command to clear all flags from all axes at once. These commands only clear the error flags, they do not clear the cause of the error.

ASCII	MTS[axis]	CLR	PS[axis]
Standalone	MTS[axis]	ECLEAR[axis]	PS[axis]

Operation procedure

There is no particular procedure.

Related commands

MST, CLR, PS

2.7 Standalone program Specification

The Commander core supports standalone programming, which allows the controller to execute a user-defined program that is stored in its internal memory. The standalone program can be run independently of USB and serial communication, or while communication is active.

2.7.1 Program Development

Commander core utilizes A-Script, a BASIC-like language that helps the programmer quickly write programs for motion control projects, including conditional statements, subroutines, and mathematical expressions. The programming syntax uses a series of standalone commands, listed in section 5.2 Standalone Command Set, that allows the programmer to build single and multithread programs that can be compiled and run in the controller. By writing lines of code using the set of commands that make up the A-Script language, set-up, control of motion, management of inputs and outputs, and error handling can all be addressed. A basic program consists of a start line (**PRGxx**) and an end line (**END**). All code is written between the PRG and END lines. Use the Utility software to write, compile, download, upload, and execute programs.

2.7.2 Conditional loop control

A-Script supports a number of conditional statements that will perform different actions for different conditions. A-Script supports the following conditional statements: **IF/ELSEIF/ELSE** loop and **WHILE** loop. Table 2-30 conditions below shows the available conditions.

Symbol	Meaning
=	Equals
!=	Not Equal
>	Greater than
<	Less than
>=	Greater than or Equals
<=	Less than or Equals

Table 2-30 conditions

Examples of using conditional loop control in a program can be found in sections 5.4.3 Standalone Example Program 3 – Single Thread, through 5.4.10 Standalone Example Program 10 – Multi-thread with subroutine.

2.7.3 Multi-thread

The Commander core utilizes four independent processors that can run simultaneously. Therefore, it is possible to run multiple programs at the same time with or without coordination.

While multiple programs may be running independently, global variable settings can cause unintended consequences between individual programs. As an example, a thread that changes the position mode to incremental could inadvertently cause the position moves of a second thread to respond incorrectly, if it was based on absolute position mode. As such it is recommended that all motion activities be written in one thread, which will prevent unintended interactions between axes, utilizing the other threads to manage I/O and other monitoring operations. 5.4.8 Standalone Example Program 8 – Multi-thread, through 5.4.10 Standalone Example Program 10 – Multi-thread with subroutine contains examples of using multi-threads in a program.

2.7.4 Subroutines

Up to 32 (0-31) subroutines can be called upon. Subroutines are independent program snippets that can be accessed based on the needs of the programmer. Typically, subroutines are used to perform functions that are repeated throughout the operation and can be used in conjunction with logic statements related to motor actions or the status of inputs and outputs. Sections 5.4.6 Standalone Example Program 6 – Single Thread with subroutine, 5.4.9 Standalone Example Program 9 – Multi-thread with subroutine, and 5.4.10 Standalone Example Program 10 – Multi-thread with subroutine contains examples of using a subroutine in a program.

Subroutine 31 is used specifically for error handling and is executed whenever an error occurs.

Note: subroutines can be shared by both standalone programs and the ASCII command set. Once a subroutine is written into the flash memory, it can be recalled via ASCII communication using the **GS** command. Standalone programs can also jump to a subroutine using the **GOSUB** command. The subroutines are referenced by their subroutine number [**SUB 0**- **SUB 31**]. If a subroutine number is not defined, the controller will return with an error.

2.7.5 Error Handling

Subroutine 31 (**SUB 31**) is used specifically for error handling and is executed whenever an error occurs. If **SUB 31** is not defined, the program will cease execution and go into an error state.

How errors are addressed can be coded in SUB 31. Conditional statements that check for specific error conditions can execute additional operations, depending on the error conditions. Typically, the code within subroutine 31 will contain the standalone command **ECLEAR[axis]** in order to clear the current error. Sections 5.4.7 Standalone Example Program 7 – Single Thread with error handling, 5.4.9 Standalone Example Program 9 – Multi-thread with subroutine, and 5.4.10 Standalone Example Program 10 – Multi-thread with subroutine contain examples of using subroutine 31 to perform error handling.

The return jump from subroutine 31 will be determined by the ASCII command **SAP**. Write a "0" to this setting to have the standalone program jump back to the last performed line. Write a "1" to this setting to have the standalone program jump back to the first line of the program.

Value	Description
0	Return to last excited line
1	Return to first line of program

Table 2-31 SAP command

2.7.6 Standalone Variables

The Commander core has 100 32-bit signed standalone variables available for general purpose use. They can be used to perform basic calculations and support integer operations. The **V[0-99]** command can be used to access the specified variables. The syntax for all available operations can be found in the table below. Note that these operations can only be performed in standalone programming.

Operator	Description	Example
+	Integer Addition	V1=V2+V3
-	Integer Subtraction	V1=V2-V3
*	Integer Multiplication	V1=V2*V3
/	Integer Division (round down)	V1=V2/V3
%	Modulus	V1=V2%5
>>	Bit Shift Right	V1=V2>>2
<<	Bit Shift Left	V1=V2<<2
&	Bitwise AND	V1=V2&7
|	Bitwise OR	V1=V2|8
~	Bitwise NOT	V1=~V2

Table 2-32 operations

All variables can be accessed and written with the ASCII **V[0-99]** command and stored to flash memory using the **STORE** command.

2.7.7 Compiling, Saving, Downloading

Standalone programs can be written to the Commander core using the Windows GUI. Refer to the Software Manual for details. Once a standalone program is written by the user, it is then compiled, saved for future reference and downloaded to the Commander core. Each line of written standalone code creates 1-4 assembly lines of code after compilation. See the software manual for more information.

2.7.8 Standalone Control

The Commander core supports the simultaneous execution of four standalone programs. All programs can be controlled by the **SACTRL[0-3]=[control]** ASCII command and the **SR[0-3]=[control]** standalone command. Program 0 will use command **SACTRL0**, Program 1 will use command **SACTRL1**, and so on. The following [control] assignments can be used for the standalone control command.

Value	Description
0	Stop standalone program
1	Start standalone program
2	Pause standalone program
3	Continue standalone program

Table 2-33 SACTRL command

2.7.9 Standalone Status

The **SASTAT[0-3]** command can be used to determine the current status of the specified standalone program. The table below details the returned values of this command.

Value	Description
0	Idle
1	Running
2	Paused
3	N/A
4	Error

Table 2-34 SASTAT command

The **SPC[0-3]** command can also be used to find the current assembled line that the specified standalone program is executing. Note that the return value of the **SPC[0-3]** command is referencing the assembly language line of code and does not directly transfer to the pre-compiled user generated code. The return value can range from [0-1799].

2.7.10 Standalone Run on Boot-Up

Standalone can be configured to run on boot-up using the **SLOAD** command. Once this command has been issued, the **STORE** command will be needed to save the setting to flash memory. It will take effect on the following power cycle. See description in the table below for the bit assignment of the **SLOAD** setting.

Bit	Description
0	Standalone Program 0
1	Standalone Program 1
2	Standalone Program 2
3	Standalone Program 3

Table 2-35 SLOAD command

ASCII	SACTRL[0-3]	SASTAT[0-3]	SPC[0-3]	GS[0-31]	V[0-99]	SLOAD
Standalone	SR[0-3]	—	—	GOSUB[0-	V[0-99]	—

ASCII	SAP	STORE
Standalone	—	STORE

3.0 Communication Interface

The Commander core has a number of communication methods available. The default method is USB communication. There are also a number of available Serial Communication methods, including UART, I^2C, and SPI.

3.1 USB Communication

The Commander core is a HID device and is USB 2.0 compliant. No additional USB drivers are needed to communicate with the Commander core. All USB communication is done using an ASCII command protocol.

3.1.1 Typical USB Setup

When using a breakout board, the Commander core can be connected to a PC directly via USB or through a USB hub, allowing for multiple units to connect and communicate at once. All USB cables should have noise suppression chokes, also known as ferrite beads, to avoid communication loss or interruption. See Figure 3-40.

When designing your own PCB please follow the layout guidelines as outlined by the USB 2.0 specifications. D+/D- traces should be the same length, maintain a nominally 90Ω differential impedance, and have a continuous ground plane directly beneath the D+/D- traces and extend at least five times the spacing width to either side of the D+/D- traces.

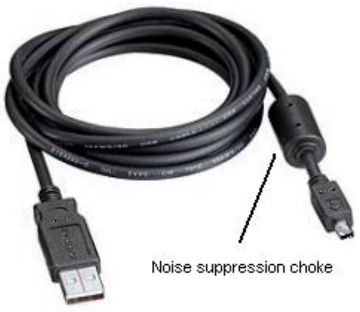


Figure 3-40

3.1.2 HID Communication API

Communication between the PC and Commander core is done using the Windows-compatible DLL API function calls shown below. Windows programming languages that use a DLL, such as Visual BASIC, Visual C++, Java, Python, LabVIEW, etc. can be used to communicate with the Commander core.

Typical communication transaction time for sending a command from the PC and getting a reply from the controller using the **SendReceiveCommanderHID** API function is in single-digit milliseconds. This value will vary with the CPU speed and type of command requested.

The DLL file **CMDHidApi.dll** (32bit) or **CMDHidApi_64.dll** (64bit) will be needed to access these API functions. These DLL files are not included with this product, please download them from our web site.

For HID communication, the following DLL API functions are provided:

Function name	Description
GetCommanderHIDnumber()	Return the number of Commander core devices connected to the host
GetCommanderHIDName(int index, char* name)	Obtain the device name string of a device connected to host
OpenCommanderHID(HANDLE* pHandle, char* name)	Obtain the HANDLE* to operate a device
CloseCommanderHID(HANDLE* pHandle)	Close the specified HANDLE*
SendReceiveCommanderHID(HANDLE* pHandle	Communication between the host and the device

Table 3-36 DLL/API functions

3.1.2.1 How to use

1. Obtain the number of Commander core HID devices connected to the PC with **GetCommanderHIDnumber**.
2. Get the device name string of the number (index) specified by **GetCommanderHIDName**.
3. Get the HANDLE* to manipulate the Commander core with the device name string specified by **OpenCommanderHID**.
4. Communicate with the Commander core using **SendReceiveCommanderHID** by specifying the acquired HANDLE*.
5. After sending the command string, the reply string will return.
    a. For the command definition and the returned serial number string, refer to the Command Reference.
6. Close the HANDLE* with **CloseCommanderHID** when finished with all operations.

3.1.2.2 GetCommanderHIDnumber

Overview

Used to get total number of Commander core HID devices connected to the host. The number of devices is represented in the return value. The function will return 0 if no devices are connected. This function shows that the Commander core is correctly recognized by the host.

Argument

None

Reply

Int type

0	No recognized device
1	One device was recognized
η	η pcs of device are recognized

Example

(1) VC

```
int numDev;                         // The number of devices
numDev = GetCommanderHIDnumber();   // Obtain the number of devices connected to the host
```

(2) VB

```
Dim numDev as Integer               ‘ The number of devices
numDev = GetCommanderHIDnumber()    ‘ Obtain the number of devices connected to the host
```

3.1.2.3 GetCommanderHIDName

Overview

Obtains the device name string of a Commander core HID device by its index number within the detected number of devices. The first detected device is indexed as 0, the second as 1, and so on. Will return 1 if successful, 0 if unsuccessful.

Argument

int number	Specify the index number of the device to get name for.
char* name	Device name,

The device name will be "**CMD-4CR-*xx***" where ***xx*** is the Identification Number (**ID**) of the device.

Reply

Int type

0	Send Receive failed
1	Send Receive success
Other (-*value*)	Host communication failed (3.1.2.7 Communication Error Codes)

Example

(1) VC

```
int result;                                   // Function result
char devName[16];                             // Device name
result = GetCommanderHIDName(0, devName);     // Obtain the device name
```

(2) VB

```
Public devName As New String("", 15)          ' Device name
Dim result as Integer                         ' Function result
result = GetCommanderHIDName(0, devName);     ' Obtain the device name
```

3.1.2.4 OpenCommanderHID

Overview

Opens a handle to a Commander core HID device by its device name string. Returns 1 if successful, 0 if unsuccessful.

Execute this for restart after disconnecting with the function, **CloseCommanderHID**.

Argument

HANDLE* pHandle	Device handle
char* name	Device name string,

The device name string is acquired by the function, **GetCommanderHIDName**

Reply

Int type

0	Send Receive failed
1	Send Receive success
Other (-*value*)	Host communication failed (3.1.2.7 Communication Error Codes)

Example

(1) VC

```
int result;                                          // Function result
HANDLE devHandle;                                    // Device handle
char devName[16];                                    // Serial number string
result = OpenCommanderHID(devHandle, devName);       // Open device, Obtain handle
```

(2) VB

```
Public devHandle As Integer                          ‘ Device handle
Public devName As New String(“”, 15)                 ‘ Serial number string
Dim result as Integer                                ‘ Function result
result = OpenCommanderHID(devHandle, devName);       ‘ Open device, Obtain handle
```

3.1.2.5 CloseCommanderHID

Overview

Closes a HANDLE* previously opened on a Commander core HID device. The handle cannot be used for communication once this function is called. Returns 1 if successful, 0 if unsuccessful.

It is recommended that you close all handles when you end the program.

Argument

HANDLE* pHandle   Device handle

The device handle is acquired by the function, **OpenCommanderHID**

Reply

Int type

0	Send Receive failed
1	Send Receive success
Other (-*value*)	Host communication failed (3.1.2.7 Communication Error Codes)

Example

(1) VC

```
int result;                                // Function result
HANDLE devHandle;                          // Device handle
result = CloseCommanderHID (devHandle);    // Close the handle
```

(2) VB

```
Public devHandle As Integer                ‘ Device handle
Dim result as Integer                      ‘ Function result
result = CloseCommanderHID (devHandle);    ‘ Close the handle
```

3.1.2.6 SendReceiveCommanderHID

Overview

Sends a command to the Commander core HID device through the open path handle which is specified in the parameter “pHandle” and reads the response from the device. The command to be sent must be passed as a string in parameter "command." The response string received from the device is stored in "reply." Returns 1 if successful, 0 if unsuccessful.

The command and reply parameters must be 63 bytes in length.

Argument

HANDLE* pHandle	Device handle

The device handle is acquired by the function, **OpenCommanderHID**

char* command	Command

Requires the storage capacity for 63 bytes

char* reply	Reply data

Requires the storage capacity for 63 bytes

Reply

Int type

0	Send Receive failed
1	Send Receive success
Other (-*value*)	Host communication failed (3.1.2.7 Communication Error Codes)

Example

(1) VC

```
int result;                                                   // Function result
HANDLE devHandle;                                             // Device handle
char cmdStr[64];                                              // Command string
char replyStr[64];                                            // Reply string


strcpy(cmdStr, “PX”);                                         // Specify a command string
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr); // Execute the communication
```

(2) VB

```
Public devHandle As Integer                                   ‘ Device handle
Dim result as Integer                                         ‘ Function result
Dim cmdStr As New String(””, 63)                              ‘ Command string
Dim replyStr As New String(””, 63)                            ‘ Reply string


cmdStr = PX                                                   ‘ Specify a command string
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr); ‘ Execute the communication
```

3.1.2.7 Communication Error Codes

Table 3-37 API error codes below is a list of possible error codes from the API and their meaning.

Returned Value	Description
-1	API to HID, timeout error
-2	API to HID, communication failed with the device
-3	Invalid function
-4	Device not found
-5	Communication path not found
-6	Communication refused by the device
-7	Invalid handle
-8	Not enough memory
-9	Out of memory
-10	In protect status
-11	Write error
-12	Read out error
-13	Invalid parameter
-14	Open failed
-15	Response was received from the device with an unsupported report ID
-16	Response of incorrect size was received from the device
-255	Unknown error during communication

Table 3-37 API error codes

3.2 Serial Communication

The Commander core has the ability to communicate over a serial communication interface using an ASCII protocol. Various serial interfaces can be used, depending on the specifications of the breakout board.

3.2.1 Communication Port Settings

The Commander core has the communication port settings shown in Table 3-38 serial port settings below.

Parameter	Setting
Byte Size	8 bits
Parity	None
Flow Control	None
Stop Bit	1

Table 3-38 serial port settings

Commander core provides the user with the ability to set the desired baud rate of the serial communication. In order to make these changes, first set the desired baud rate by using the **DB** command. By default, the Commander core has a baud rate setting of 9600 bps.

Return Value	Description
1	9600
2	19200
3	38400
4	57600
5	115200

Table 3-39 DB command

To write the values to the device’s flash memory, use the **STORE** command. The new baud rate will now be written to memory, but will not go into effect until after a complete power cycle.

3.2.2 ASCII Protocol

The following ASCII protocol should be used for sending commands and receiving replies from the Commander core. For details on valid ASCII commands refer to Section 4.0 ASCII Language Specification or the Command Reference.

Sending Command

ASCII command string in the format of:

@[DeviceName][ASCII Command][CR]   ‘[CR] character has ASCII code 13

Receiving Reply

The response will be in the format of:

[Response][CR]   ‘[CR] character has ASCII code 13

Example

For querying the polarity of the X-axis

```
Send:   @01POLX[CR]
Reply:  7[CR]
```

For reading the digital input status

```
Send:   @01DI[CR]
Reply:  8[CR]
```

For performing a store to flash memory

```
Send:   @01STORE[CR]
Reply:  OK[CR]
```

3.3 Identification Number

The Commander core allows for each device to have a unique identification number, allowing up to 64 devices to be connected to the same host. If multiple devices are connected to the host, the identification number for each device should be unique.

By default, all Commander cores are shipped from the factory with identification number 01. The current identification number of a device can be found by reading the **ID** command. Also, the device name will always include the current identification number. For USB communication the device name will be "CMD-4CR-xx" where xx is the identification number of the device. For serial communication only the identification number of the device is used.

In order to change the identification number of the device, first store the desired number using the **ID** command. Note that this value must be within the range [00-99].

For example, the command **ID=02** can be sent to change the identification number to 02. To save a modified identification number to the flash memory of the device, use the **STORE** command. The new identification number will not take effect until after a power cycle.

3.4 Windows GUI

A Windows GUI program is available for the Commander core via download from our website. The Windows GUI serves two main functions: (1) provides a shortcut to every ASCII command to assist in testing and troubleshooting the setup of the Commander core and your own interface code, and (2) allows you to test, program, compile, download and debug the controller using the A-script standalone language. See the CMD-4CR Software Manual for details.

4.0 ASCII Language Specification

Important Note: All the commands described in this section are interactive ASCII commands and are not analogous to standalone commands. Refer to Section 5.0 Standalone Language Specification.

For more detailed information on any command, please refer to the command refence.

Invalid commands are returned with a "?" Always check for a proper reply when a command is sent. For USB and RS-485 communication, the commands detailed in section 4.1 ASCII Command Set are valid.

4.1 ASCII Command Set

Command	R/W	Description	Reference Chapter
ABORT	W	Immediately stops all motion for all axes	2.2.3
ABORT[axis]	W	Immediately stops motion in [axis]	
ABS	W	Turns on absolute coordinate mode	2.2.4.1
ACC	R	Returns current global acceleration value in ms	2.1.4
ACC=[value]	W	Sets global acceleration value in ms	
ACC[axis]	R	Returns current [axis] acceleration value in ms	
ACC[axis]=[value]	W	Sets [axis] acceleration value in ms	
AI[ch]	R	Returns analog input status for [ch] in mV	2.1.3.3.2
ARC[A1][A2]N[C1]:[C2]:[Θ]	W	Arc interpolation with axes [A1], [A2], centers at [C1], [C2], CCW direction to [Θ]	2.3.2.1
ARC[A1][A2]P[C1]:[C2]:[Θ]	W	Arc interpolation with axes [A1], [A2], centers at [C1], [C2], CW direction to [Θ]	
ARCTN[C1]:[C2]:[Θ]:[target]	W	XY Arc interpolation, centers at [C1], [C2], CCW direction to [Θ] with Z linear interpolation to [target]	2.3.3
ARCTP[C1]:[C2]:[Θ]:[target]	W	XY Arc interpolation, centers at [C1], [C2], CW direction to [Θ] with Z linear interpolation to [target]	
BF	W	Disable buffered move	2.3.4
BO	W	Enable buffered move	2.3.4
BSTART	W	Start processing the buffer (buffer must be enabled)	2.3.4
BSTAT	R	Status of the buffer	2.3.4
CIR[A1][A2]N[C1]:[C2]	W	Circular interpolation with axes [A1], [A2], centers at [C1], [C2], CCW direction	2.3.2
CIR[A1][A2]P[C1]:[C2]	W	Circular interpolation with axes [A1], [A2], centers at [C1], [C2], CW direction	
CIRTN[C1]:[C2]:[target]	W	XY circular interpolation, centers at [C1], [C2], CCW direction with Z linear interpolation to [target]	2.3.3
CIRTP[C1]:[C2]:[target]	W	XY circular interpolation, centers at [C1], [C2], CW direction with Z linear interpolation to [target]	
CLR[axis]	W	Clears limit, alarm status bit, and StepNLoop errors for [axis]	2.1.3.1.1.1 2.1.3.1.1.2 2.1.3.1.2 2.5.3 2.6.2.3

Command	R/W	Description	Reference Chapter
D[axis]	R	Returns StepNLoop value for [axis]	2.5.3 2.6.1.3
DB	R	Returns baud rate	2.1.7
DB=[value]	W	Sets baud rate to [value]	
DEC	R	Returns the current global deceleration value in ms	2.1.4
DEC=[value]	W	Sets global deceleration value in ms	
DEC[axis]	R	Returns current [axis] deceleration value in ms	2.1.4
DEC[axis]=[value]	W	Sets [axis] deceleration value in ms	
DI	R	Returns 4-bits of general-purpose digital inputs	2.1.3.2.1
DI[in]	R	Returns bit status of general digital input [in]	2.1.3.2.1
DIP	R	Returns the digital input polarity	2.1.3.2.1 2.1.6.3 2.1.7
DIP=[pol]	W	Sets the digital input polarity [pol]	
DO	R	Returns 4-bits of general-purpose digital outputs.	2.1.3.2.2
DO=[value]	W	Sets 4-bits of general-purpose digital outputs.	
DO[out]	R	Returns bit status of general digital output [out]	2.1.3.2.2
DO[out]=[value]	W	Sets bit of general-purpose digital output [out]	
DOBOOT	R	Returns the DO boot-up state.	2.1.3.2.2 2.1.5 2.1.7
DOBOOT=[value]	W	Sets the DO boot-up state.	
DOP	R	Returns the digital output polarity	2.1.3.2.2 2.1.6.3 2.1.7
DOP=[pol]	W	Sets the digital output polarity [pol]	
E[axis]	R	Returns the encoder value of [axis]	2.6.1.2
E[axis]=[value]	W	Sets the encoder value of [axis]	
EINT	R	Returns the interpolation setting	2.1.7 2.2.4.2 2.3.1
EINT=[value]	W	Enables/disables interpolation	
EO	R	Returns 4-bits of enable output value	2.1.2.1 2.1.2.2 2.2.1
EO=[value]	W	Sets 4-bits of enable output value	
EO[en]	R	Returns bit of enable output [en]	
EO[en]=[value]	W	Sets bit of enable output [en]	
EOBOOT	R	Returns EO boot-up state	2.1.5 2.1.7 2.2.1
EOBOOT=[value]	W	Sets EO boot-up state	
EP	R	Returns the encoder value for all 4 axes	2.6.1.2
ERC[axis]	R	Returns the enable status of the deflection counter clear output	2.1.2.2 2.1.3.1.1.1 2.1.6 2.1.6.1 2.1.7 2.2.5
ERC[axis]=[value]	W	Enables/disables the deflection counter clear output	
ERCD	R	Returns the pulse delay set value	2.1.6

Command	R/W	Description	Reference Chapter
ERCD=[value]	W	Sets the pulse delay value (2-bits)	2.1.6.1 2.1.7 2.1.2.2
ERCP	R	Returns the pulse width set value	2.1.6 2.1.6.1 2.1.7 2.1.2.2
ERCP=[value]	W	Sets the pulse width value (3-bits)	
ESTOP	W	Performs an emergency stop (same as triggering CEMG)	2.1.3.1.2
EXST	R	Returns the status of the external start feature	2.1.1.2 2.1.6 2.1.6.2 2.1.7
EXST=[value]	W	Enable/disable the external start feature	
GS[sub]	W	Call a defined subroutine [sub] (value 0 – 31)	2.7.4
H[axis][+/-][mode]	W	Home [axis] in positive or negative direction using homing mode [mode] (see Appendix A for detailed information on homing modes)	2.2.5
HSPD	R	Returns the global high speed setting	2.1.4
HSPD[value]	W	Sets the global high speed value	
HSPD[axis]	R	Returns the high speed setting for [axis]	2.1.4
HSPD[axis]=[value]	W	Sets the high speed value for [axis]	
I[X target]:[Y target]:[Z target]:[speed]	W	Writes an XYZ interpolated move to buffer. Sets each axis move target and desired move speed.	2.3.4
ID	R	Returns the Commander's ID	2.1.1.1 2.1.7
ID=[value]	W	Sets the Commander's ID. Range is 00 to 99. Use STORE command to save.	
IERR	R	Returns the ignore limit/alarm error status	2.1.3.1.1.1 2.1.3.1.2 2.1.6 2.1.7
IERR=[value]	W	Sets the ignore limit/alarm status	
INC	W	Turns on incremental coordinate mode	2.2.4.1
INP[axis]	R	Returns enable/disable status of positioning complete signal for [axis]	2.1.2.2 2.1.6 2.1.7
INP[axis]=[value]	W	Enables/disables the positioning complete signal for [axis]	
IO	R	Returns the status of all configurable I/O ports (32-bit)	2.1.3.3.1
IO=[value]	W	Sets the bit status of all configurable outputs	
IO[port]	R	Returns the status of configurable I/O at [port]	2.1.3.3.1
IO[port]={0,1}	W	Sets the value of configurable output bit at [port]	
IOBOOT	R	Returns the status of all general purpose I/O at startup (32-bit)	2.1.3.3.1 2.1.5 2.1.7
IOBOOT=[value]	W	Sets the value of all configurable outputs at startup	
IOCFG	R	Returns the current configuration of the configurable I/O ports	2.1.3.3.1 2.1.5

Command	R/W	Description	Reference Chapter
IOCFG=[value]	W	Sets the configuration of the configurable I/O ports	2.1.7
IOP	R	Returns the configurable I/O polarity setting (2-bit logic)	2.1.3.3.1 2.1.6.3 2.1.7
IOP=[0,1]	W	Sets the configurable I/O polarity (2-bit logic)	
J[axis][+/-]	W	Jogs the motor in the [+] or [-] direction	2.2.2
JDEL[axis]	R	Returns the maximum allowable speed change with joystick control for [axis]	2.1.3.3.2 2.1.7 2.5.1.2
JDEL[axis]=[value]	W	Sets the maximum allowable speed change with joystick control for [axis]	
JENA	R	Returns status for joystick control	2.1.3.3.2 2.1.7 2.5.1
JENA=[value]	W	Enables/disables joystick control	
JLIM[number]	R	Returns the current operating behavior limit setting of joystick operation	2.1.3.3.2 2.1.7 2.5.1.3
JLIM[number]=[value]	W	Sets the operating behavior limit value of joystick operation	
JMAX[axis]	R	Returns the maximum voltage setting of the input terminal for joystick operation of [axis]	2.1.3.3.2 2.1.7 2.5.1.1
JMAX[axis]=[value]	W	Sets the maximum terminal voltage of [axis] for joystick operation	
JMIN[axis]	R	Returns the minimum voltage setting of the input terminal for joystick operation of [axis]	2.1.3.3.2 2.1.7 2.5.1.1
JMIN[axis]=[value]	W	Sets the minimum terminal voltage of [axis] for joystick operation	
JSPD[axis]	R	Returns the maximum speed setting of [axis] for joystick operations	2.1.3.3.2 2.1.7 2.5.1.2
JSPD[axis]=[value]	W	Sets the maximum speed of [axis] for joystick operations	
JTOL[axis]	R	Returns the current dead band around mid-voltage range of [axis] for joystick operation	2.1.3.3.2 2.1.7 2.5.1.1
JTOL[axis]=[value]	W	Sets the dead band range around mid-voltage of [axis] for joystick operation	
LSPD	R	Returns global low speed setting	2.1.4
LSPD=[value]	W	Sets global low speed value	
LSPD[axis]	R	Returns low speed setting for [axis]	2.1.4
LSPD[axis]=[value]	W	Sets low speed value for [axis]	
LT[axis]=[value]	W	Enables/disables latch feature	2.1.3.2.1
LTE[axis]	R	Returns the previously latched encoder position	2.1.3.2.1
LTP[axis}	R	Returns the previously latched step position	2.1.3.2.1
LTS[axis]	R	Returns the current latch status	2.1.3.2.1
MM	R	Returns the current move mode for moves (absolute or incremental)	2.2.4.1
MP[axis]	R	Returns the current manual pulse generator position	2.1.3.1.4 2.5.2.2
MP[axis]=[values]	W	Sets the manual pulse generator position	
MPD[axis]	R	Returns the divider parameter for manual pulse generator operation	2.1.3.1.4 2.1.7

Command	R/W	Description	Reference Chapter
MPD[axis]=[value]	W	Sets the divider parameter for manual pulse generator operation	2.5.2 2.5.2.2
MPE[axis]	R	Returns the manual pulse generator enable status	2.1.3.1.4 2.1.7 2.5.2
MPE[axis]=[value]	W	Enable/disable the manual pulse generator	
MPM[axis]	R	Returns the multiplier parameter for manual pulse generator operation	2.1.3.1.4 2.1.7 2.5.2 2.5.2.2
MPM[axis]=[value]	W	Sets the multiplier parameter for manual pulse generator operation	
MST[axis]	R	Returns all motor status, buffer move status, and move mode status for [axis]	2.1.3.1.1.1 2.1.3.1.1.2 2.1.3.1.2 2.6.2.1
P[axis]	R	Returns the position value for [axis]	2.6.1.1
P[axis]=[value]	W	Sets the position value for [axis]	
POL[axis]	R	Returns polarity parameter for [axis] (see Table 6.9)	2.1.2.1 2.1.2.2 2.1.2.3 2.1.3.1.1.2 2.1.6 2.1.6.3 2.1.7 2.2.1 2.5.2 2.5.2.1
POL[axis]=[pol]	W	Sets the polarity parameter for [axis]	
PP	R	Returns current pulse counter value of all 4 axes	2.6.1.1
PS[axis]	R	Returns the current pulse speed for [axis]	2.6.2.2
PWM[1-2]	R	Returns current PWM duty cycle setting (in %)	2.1.3.3.3
PWM[1-2]=[value]	W	Set the PWM duty cycle (in %)	
SACTRL[prgNo]=[0-3]	W	Controls the standalone [prgNo]: stop(0), start(1), pause(2), continue(3)	2.7.8
SAP	R	Returns the return jump line (standalone error handling)	2.7.5
SAP[0-1]	W	Sets the return jump line (standalone error handling)	
SASTAT[0-3]	R	Returns status for standalone program.	2.7.9
SCV[axis]	R	Returns S-curve control status for [axis]	2.1.4 2.1.6
SCV[axis]=[0,1]	W	Enable/disable S-curve acceleration for [axis]. Trapezoidal accel/decel used if disabled.	
SDC[axis]	R	Returns the configuration of the Slow Down decel operation for [axis]	2.1.3.1.1.2 2.1.6 2.1.7
SDC[axis]=[value]	W	Sets the configuration for the Slow Down decel (decel only or decel and stop) for [axis]	
SDE[axis]	R	Status of the Slow Down decel operation for [axis]	2.1.3.1.1.2 2.1.6 2.1.7
SDE[axis]=[value]	W	Enable/disable the external deceleration feature for [axis]	
SL[axis]	R	Returns status of StepNLoop enable for [axis]	2.1.7

Command	R/W	Description	Reference Chapter
SL[axis]=[value]	W	Enable/Disable StepNLoop feature for [axis]	2.5.3
SLA[axis]	R	Returns maximum attempts to complete StepNLoop operation for [axis]	2.1.7 2.5.3
SLA[axis]=[value]	W	Sets maximum attempts to complete StepNLoop operation for [axis]	
SLE[axis]	R	Returns error range of StepNLoop operation for [axis]	2.1.7 2.5.3
SLE[axis]=[value]	W	Sets error range of StepNLoop operation for [axis]	
SLOAD[prg#]	R	Returns the RunOnBoot parameter for [prg#]	2.1.5 2.1.7 2.7.10
SLOAD[prg#]=[value]	W	Sets the RunOnBoot parameter for [prg#]	
SLR[axis]	R	Returns the StepNLoop ratio for [axis] (ppr/cpr)	2.1.7 2.5.3
SLR[axis]=[value]	W	Sets the StepNLoop ratio for [axis] (ppr/cpr)	
SLS[axis]	R	Returns StepNLoop status for [axis]	2.5.3
SLT[axis]	R	Returns StepNLoop tolerance for [axis]	2.1.7 2.5.3
SLT[axis]=[value]	W	Sets StepNLoop tolerance for [axis]	
SPC[prg#]	R	Returns program counter for [prg#]	2.7.9
SSPD[axis]=[value]	W	Performs on-the-fly speed change for [axis]. Buffer must be disabled	2.4.1
SSPDM[axis]	R	Returns the current on-the-fly speed mode	2.4.1
SSPDM[axis]=[value]	W	Sets the on-the-fly speed mode	
STOP	W	Performs ramp down to low speed and stop if motor is moving for all axes	2.2.3
STOP[axis]	W	Performs ramp down to low speed and stop if motor is moving for [axis]	
STORE	W	Saves parameters to flash memory	2.1.1.1 2.1.3.2.2 2.1.3.3.1 2.1.5 2.1.6 2.1.7 2.2.1 2.7.6
SYNC[axis]	R	Returns SYNC condition and source	2.1.3.2.2
SYNC[axis]=[value]	W	Sets SYNC condition and source [axis]	
SYNF[axis]	W	Disable synchronization window and synchronization output	2.1.3.2.2
SYNMAX[axis]	R	Returns the current maximum value of the Sync Pulse Window for [axis]	2.1.3.2.2
SYNMAX[axis]=[value]	W	Sets the maximum value of the Sync Pulse Window for [axis]	
SYNMIN[axis]	R	Returns the current minimum value of the Sync Pulse Window for [axis]	2.1.3.2.2
SYNMIN[axis]=[value]	W	Sets the minimum value of the Sync Pulse Window for [axis]	
SYNO[axis]	W	Enable synchronization output	2.1.3.2.2

Command	R/W	Description	Reference Chapter
SYNP[axis]	R	Returns the synchronization position value or distance between pulses for continuous sync.	2.1.3.2.2
SYNP[axis]=[value]	W	Sets the synchronization position value or distance between pulses for continuous sync.	
SYNS[axis]	R	Returns the status of the synchronization feature	2.1.3.2.2
SYNWF[axis]	W	Disables the synchronization window for [axis]	2.1.3.2.2
SYNWO[axis]	W	Enables the synchronization window and synchronization output feature for [axis]	2.1.3.2.2
T[axis][newtarget]	W	Sets the on-the-fly target position change for [axis]	2.4.2
V[var]	R	Returns the standalone [var] value	2.1.7 2.7.6
V[var]=[value]	W	Sets the standalone [var] value	
VER	R	Returns the current firmware version	2.1.1.1
[axis][targetpos]	W	Individual move command for [axis] to [targetpos]	2.2.4.2
X[targetX]Y[targetY]Z[target Z]U[targetU]	W	Interpolation move command (include all axes that will be interpolated). EINT must be enabled (1)	2.2.4.2 2.3.1
ZCNT[axis]	R	Returns the number of Z-index pulses to count for [axis]. Used for relevant moves (ZMOVE and homing routines)	2.1.7 2.2.5.1
ZCNT[axis]=[value]	W	Sets the Z-index count to be used for relevant moves (ZMOVE and homing routines). Range is 1 – 16.	
ZMOVE[axis][+/-]	W	Perform the EZ Count operation for [axis] in positive (+) or negative (-) direction	2.2.5.1

Table 4-40 ASCII commands

4.2 Error Codes

If an ASCII command cannot be processed by the Commander core, the controller will reply with an error code. See below for possible error responses:

Error Code	Description
?Unknown Command: [Command]	The ASCII command is not understood by the Commander core.
?Error Moving	The move command could not be processed. One or more of the required axes may be in error state or in motion.
?Error Alarm	The move command could not be processed. The Alarm signal is in the on state.
?Error +Limit	The move command could not be processed. The + End Limit is in the on state.
?Error -Limit	The move command could not be processed. The - End Limit is in the on state.
?Error ESTOP	The move command could not be processed. The EMG signal is in the on state.
?Error SNL	The move command could not be processed. The StepNLoop control is in error.
?Invalid Index	The Index number is invalid
?Program Error	The command cannot be processed because a standalone program is in error state.
?Sub not Initialized	Call to a subroutine using the GS command is not valid because the specified subroutine has not been defined.
?Buffer Full	Move cannot be added to a buffer register because all registers are full.
?Buffer Disabled	Buffer command is not valid because buffer mode is disabled.
?Buffer Enabled	SSPD command could not be processed because buffer mode is enabled.
?SSPD Error	SSPD command cannot be processed because the SSPD mode is not defined or out of range. S-curve should also be disabled.
?Override Failed	Override operation failed

Table 4-41

4.3 Examples of command control from a PC

The following examples are provided to assist your understanding. Although they were produced with the utmost care, if you find any points that are unclear, wrong, or have inadequate descriptions, please let us know.

For more examples please refer to the command refence.

4.3.1 Start and stop of jog move

Summary of operation

Set the speed of the X-axis movement and turn on excitation. After that, start the movement of the X-axis in the negative direction. After a while, decelerate and stop the axis.

Program code

```
cmdStr = "LSPDX=500"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)     'Starting speed of X-axis set to 500 pps
cmdStr = "HSPDX=5000"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)     'Moving speed of X-axis set to 5,000 pps
cmdStr = "ACCX=200"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)     'Accel time of X-axis set to 200 ms
cmdStr = "DECX=200"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)     'Decel time of X-axis set to 200 ms
cmdStr = "EO1=1"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)     'Enable X-axis
cmdStr = "JX−"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)     ’Start moving in the (−) direction
-------------------------------------------------------------------------------------------

cmdStr = "STOPX"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)     'Decelerate and stop
```

4.3.2 Position Move

Summary of operation

Set the global moving speed and turn on the excitations of X- and Y-axes. Set positioning coordinates to absolute mode. Move the X-axis to position −20,000 and Y-axis to position 15,000.

Program code

```
cmdStr = "LSPD=500"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)    ‘Starting speed set to 500 pps (Global)
cmdStr = "HSPD=5000"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)    ‘Moving speed set to 5,000 pps (Global)
cmdStr = "ACC=200"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)    ‘Accel time set to 200 ms (Global)
cmdStr = "DEC=200"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)    ‘Deceleration time set to 200 ms (Global)
cmdStr = "EO=3"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)    ‘Enable X and Y-axis
cmdStr = "ABS"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)    ‘Set position coordinate mode to Absolute
cmdStr = "X−20000"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)    ’Start moving the X-axis to the position of
                                                                 −20,000
cmdStr = "Y15000"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)    ’Start moving the Y-axis to the position of
                                                                 15,000.
```

4.3.3 Obtaining and clearing the counter value

Summary of operation

Read and display the command pulse and feedback pulse counter on the X-axis. The counter value can be cleared only when the axis is normally stopped (stopped without error). Read the counter value in a timer event.

Program code

```
cmdStr = "PX"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)    ’Obtain the command pulse counter value on X-axis
OutplsX.Text = replyStr                                          ’Display the command pulse counter value
cmdStr = "EX"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)    ’Obtain the feedback pulse counter value on X-axis
FeedbackX.Text = replyStr                                        ‘Display the feedback counter value
cmdStr = "MSTX"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)    ’Obtain the status value of X-axis
mStatusX.Text = replyStr                                         ‘Display the status value for X-axis
```

Execute the above process repeatedly in the timer event

```
If mStatusX.Text = "0" Then                                      ‘Does X-axis stop normally?
    cmdStr = "PX=0"
    result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr) ’X-axis command pulse counter clear
    cmdStr = "EX=0"
    result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr) ’X-axis feedback pulse counter clear
End If
```

4.3.4 Homing Operation

Summary of operation

Set the Y-axis moving speed and enable axis. Then, start the Y-axis homing operation in the negative direction using homing mode 1.

Program Code

```
cmdStr = "LSPDY=500"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)      ‘Starting speed of Y-axis set to 500 pps
cmdStr = "HSPDY=5000"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)      ‘Moving speed of Y-axis set to 5,000 pps
cmdStr = "ACCY=200"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)      ‘Accel time of Y-axis set to 200 ms
cmdStr = "DECY=200"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)      ‘Decel time of Y axis set to 200 ms
cmdStr = "EO2=1"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr       ‘Enable Y-axis
cmdStr = "H−1"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)      ’Start the homing operation of Y-axis
```

4.3.5 Linear interpolation operation

Summary of operation

Set global moving speed and enable the X- and Y-axes. Set the positioning coordinate mode to absolute. Execute the linear interpolation operation, moving the X-axis to position −20,000 and the Y-axis to position 15,000.

Program code

```
cmdStr = "LSPD=500"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)    ‘Starting speed set to 500 pps (Global)
cmdStr = "HSPD=5000"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)    ‘Moving speed set to 5,000 pps (Global)
cmdStr = "ACC=200"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)    ‘Accel time set to 200 ms (Global)
cmdStr = "DEC=200"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)    ‘Decel time set to 200 ms (Global)
cmdStr = "EO=3"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)    ‘Enable X and Y-axes
cmdStr = "ABS"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)    ‘Set position coordinate mode to absolute
cmdStr = "EINT=1"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)    ‘Interpolation operation enabled
cmdStr = "X−20000Y15000"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)    ’ Start interpolation operation of X and Y-axes
```

4.3.6 Circular Interpolation Operation

Summary of operation

Set global moving speed and enable the X- and Y-axes. Set positioning coordinate mode to absolute. Execute a circular interpolation move. The center coordinates of circle are (1000, 0) and the rotation direction clockwise.

Program code

```
cmdStr = "LSPD=500"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)      ‘Starting speed set to 500 pps (Global)
cmdStr = "HSPD=5000"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)      ‘Moving speed set to 5,000 pps (Global)
cmdStr = "ACC=200"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)      ‘Accel time set to 200 ms (Global)
cmdStr = "DEC=200"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)      ‘Decel time set to 200 ms (Global)
cmdStr = "EO=3"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)      ’Enable X- and Y-axes
cmdStr = "ABS"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)      ‘Set position coordinate mode to absolute
cmdStr = "EINT=1"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)      ’Interpolation operation enabled
cmdStr = "CIRXYP1000:0"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)      ’Start interpolation operation of X and Y-
                                                                   axes
```

4.3.7 On-the-fly speed change

Summary of operation

Change the X-axis speed at 50,000 pps to 70,000 pps during motion.

Program code

```
cmdStr = "LSPDX=1000"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)      ‘Starting speed of X-axis set to 500 pps
cmdStr = "HSPDX=50000"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)      ‘Moving speed of X-axis set to 50,000 pps

                                        ---omitted steps---

cmdStr = "JX–"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)      ’Start moving in the – direction

cmdStr = "SSPDM=2"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)      ‘Double the speed range
cmdStr = "SSPD=70000"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)      ‘Update speed to 70,000 pps
```

4.3.8 Synchronization Function

Summary of operation

When the X-axis moves to position 5,000, turn on synchronization signal DO1/SYNCx.

Program code

```
cmdStr = "LSPDX=100"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)     ‘Starting speed of X-axis set to 100 pps
cmdStr = "HSPDX=1000"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)     ‘Moving speed of X-axis set to 1,000 pps

                                        ---omitted steps---

cmdStr = "SYNCX=1"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)     ‘Set X-axis sync condition
cmdStr = "SYNPX=5000"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)     ‘Set X-axis comparison data value to 5,000
cmdStr = "SYNO"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)     ‘Sync output configuration enabled
cmdStr = "X5000"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)     ‘Move X-axis to position 5,0
                                                                  ‘DO1/SYNCx will be on when movement is
                                                                  completed
```

4.3.9 Control of general purpose input/output

Summary of operation

Turn ON/OFF general-purpose output 1 in synchronization with general-purpose input 1.

Program code

```
cmdStr = "IO5"
result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)      'Read status of general-purpose input 1
If replyStr = "1" Then
    cmdStr = "IO19=1"                                              'It is ON
    result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)
Else
    cmdStr = "IO19=0"                                              'It is not ON
    result = SendReceiveCommanderHID(devHandle, cmdStr, replyStr)
End If
```

Note: The state of the general-purpose input must be read repeatedly

5.0 Standalone Language Specification

Important Note: All the commands described in this section are standalone language commands and are not analogous to ASCII commands. Refer to Section 4.0 ASCII Language Specification for details regarding ASCII commands.

For more detailed information on any command please refer to the command refence.

5.1 Using A-Script programming interface

5.1.1 Program Development

A-Script is a BASIC-like language that can help the programmer quickly write programs for motion control projects, including conditional statements (IF, ELSE, ELSEIF, ENDIF, WHILE, ENDWHILE), subroutines, and mathematical and logical expressions (=, !=, <, >, <=, >=, +, -, *, /). The programming syntax uses a series of standalone commands (listed later in this section) that allow the programmer to build single and multithread programs that can be compiled and run in the controller. By writing lines of code using the set of commands, set-up, control of motion, management of inputs and outputs, and error handling can all be addressed. A basic program consists of a start line (PRG xx) and an end line (END). All code is written between the PRG and END lines. Use the utility software to write, compile, download, upload, and execute programs.

5.1.2 Subroutines

Subroutines are program snippets that can be accessed based on the needs of the programmer and are used in conjunction with logic statements related to motor actions or status of inputs and outputs. Up to 32 (0-31) subroutines can be called upon. See section 2.7.4 Subroutines for more information.

5.1.3 Error Handling

Subroutine 31 is used for error handling. See section 2.7.5 Error Handling for more information.

5.1.4 Multi-thread

The Commander core utilizes four independent processors that can run simultaneously, making it possible to run multiple programs at the same time with or without coordination. See section 2.7.3 Multi-thread for more information.

5.1.5 Compiling, Saving, Downloading

Standalone programs can be written to the Commander core using the Windows GUI. Refer to the Software Manual for details. Once a standalone program is written by the user, it is then compiled, saved for future reference and downloaded to the Commander core. Each line of written standalone code creates 1-4 assembly lines of code after compilation. See the software manual for more information.

5.2 Standalone Command Set

Command	R/W	Description	Reference Chapter
;	-	Comments out any text following semicolon in the same line	
ABORT	W	Immediately stops all motion for all axes	2.2.3
ABORT[axis]	W	Immediately stops motion in [axis]	
ABS	W	Turns on absolute position coordinate mode	2.2.4.1
ACC	R/W	Returns/sets global acceleration value in ms	2.1.4
ACC[axis]	R/W	Returns/sets [axis] acceleration value in ms	
AI[ch]	R	Returns analog input status for [ch] in mV	2.1.3.3.2
ARC[A1][A2]N[C1]:[C2]:[Θ]	W	Arc interpolation with axes [A1], [A2], centers at [C1], [C2], CCW direction to [Θ]	2.3.2.1
ARC[A1][A2]P[C1]:[C2]:[Θ]	W	Arc interpolation with axes [A1], [A2], centers at [C1], [C2], CW direction to [Θ]	
ARCTN[Cx]:[Cy]:[θ]:[pos]	W	XY arc interpolation, centers at [C1], [C2], CCW direction to [Θ] with Z linear interpolation to [pos]	2.3.3
ARCTP[Cx]:[Cy]:[θ]:[pos]	W	XY arc interpolation, centers at [C1], [C2], CW direction to [Θ] with Z linear interpolation to [pos]	
BUFOFF	W	Disable buffered move	2.3.4
BUFON	W	Enable buffered move	2.3.4
CIR[A1][A2]N[C1]:[C2]	W	Circular interpolation with axes [A1], [A2], centers at [C1], [C2], CCW direction	2.3.2
CIR[A1][A2]P[C1]:[C2]	W	Circular interpolation with axes [A1], [A2], centers at [C1], [C2], CW direction	
CIRTN[C1]:[C2]:[pos]	W	XY circular interpolation, centers at [C1], [C2], CCW direction with Z linear interpolation to [pos]	2.3.3
CIRTP[C1]:[C2]:[pos]	W	XY circular interpolation, centers at [C1], [C2], CW direction with Z linear interpolation to [pos]	
DEC	R/W	Returns/sets the global deceleration value in ms	2.1.4
DEC[axis]	R/W	Returns/sets [axis] deceleration value in ms	
DELAY	W	Sets delay in ms	
DI	R	Return status of digital inputs. 4-bit	2.1.3.2.1
DI[1-4]	R	Get individual bit status of digital inputs. Will return [0,1].	2.1.3.2.1
DO	R/W	Returns/sets 4-bits of general-purpose digital outputs	2.1.3.2.2
DO[1-4]	R/W	Returns/sets bit status of general digital output [1-4]	2.1.3.2.2
E[axis]	R/W	Returns/sets the encoder value of [axis]	2.6.1.2

Command	R/W	Description	Reference Chapter
ECLEAR[axis]	W	Clear any motor status errors.	2.1.3.1.1.1 2.1.3.1.1.2 2.1.3.1.2 2.6.2.3 2.7.5
EINT	R/W	Returns/sets the interpolation Enable/Disable status	2.1.7 2.2.4.2 2.3.1
EO	R/W	Returns/sets 4-bits of enable output value for all axes	2.1.2.1 2.1.2.2 2.2.1
EO[1-4]	R/W	Returns/sets of enable output value for [1-4]	
GOSUB[0-31]	-	Executes subroutine [0-31]	2.7.4
HOME[axis][+/-][mode]	W	Home [axis] in positive or negative [+/-] direction using homing mode [mode] (see Appendix A: Homing Mode Actions)	2.2.5
HSPD	R/W	Returns/sets the global high-speed setting	2.1.4
HSPD[axis]	R/W	Returns/sets the high-speed setting for [axis]	

IF
ELSEIF
ELSE
ENDIF
	-	Executes the instruction following the IF statement when condition is true. Execute the instruction following the next ELSE or ELSEIF statement when the condition is false. When all conditions are false, continue executing program after ENDIF. The following operands are used in the conditional IF statement: =, >, <, > =, <=, !=	2.7.2

INC	W	Turns on incremental coordinate mode	2.2.4.1
IO	R/W	Returns/sets the status of all configurable I/O ports (32-bit)	2.1.3.3.1
IO[index]	R/W	Returns/sets the status of configurable I/O at [index]	2.1.3.3.1
ISTART	W	Start processing the buffer (buffer must be enabled)	2.3.4
JOG[axis][+/-]	W	Jogs the [axis] in the [+] or [-] direction using the HSPD setting	2.2.2
JOYDEL[axis]	W	Returns the maximum allowable speed change with joystick control for [axis]	2.1.3.3.2 2.5.1.2
JOYENA	R/W	Returns/sets the status for joystick control	2.1.3.3.2 2.5.1
JOYHS[axis]	R/W	Returns/sets the maximum speed setting of [axis] for joystick operations	2.1.3.3.2 2.5.1.2
LSPD	R/W	Returns/sets global low speed setting	2.1.4
LSPD[axis]	R/W	Returns/sets low speed setting for [axis]	2.1.4
MST[axis]	R	Returns all motor status, buffer move status, and move mode status for [axis]	2.1.3.1.1.1 2.1.3.1.1.2 2.1.3.1.2 2.6.2.1
P[axis]	R/W	Returns/sets the position value for [axis]	2.6.1.1

Command	R/W	Description	Reference Chapter
PRG[0-3] END	-	Defines the first and last line of a standalone program [0-3].	2.7.1
PS[axis]	R	Returns the current pulse speed for [axis]	2.6.2.2
SL[axis]	R/W	Returns/Sets enable/disable of StepNLoop for [axis]	2.5.3
SLS[axis]	R	Returns StepNLoop status for [axis]	2.5.3
SR[0-3]	W	Controls standalone program [0-3]. Use command to stop, start, pause, continue.	2.7.8
SSPD[axis]	W	Performs on-the-fly speed change for [axis]. Buffer must be disabled.	2.4.1
SSPDM[axis]	W	Sets the on-the-fly speed mode for [axis]	2.4.1
STOP	W	Performs ramp down to low speed and stop if motor is moving for all axes	2.2.3
STOP[axis]	W	Performs ramp down to low speed and stop if motor is moving for [axis]	
STORE	W	Saves parameters to flash memory	2.1.1.1 2.1.3.2.2 2.1.3.3.1 2.1.5 2.1.6 2.1.7 2.2.1 2.7.6
SUB[0-31] ENDSUB	-	Used to define the first and last line of a Subroutine [0-31]	2.7.4
SYNCFG[axis]	R/W	Returns/sets SYNC condition and source [axis]	2.1.3.2.2
SYNOFF[axis]	W	Disable synchronization window and output for [axis]	2.1.3.2.2
SYNON[axis]	W	Enable synchronization output for [axis]	2.1.3.2.2
SYNPOS[axis]	R/W	Sets/Returns the synchronization position value or distance between pulses for continuous sync of [axis].	2.1.3.2.2
SYNSTAT[axis]	W	Returns the status of the synchronization feature for [axis]	2.1.3.2.2
V[index]	R/W	Returns/sets the variable [index] value	2.7.6
WAIT[axis]	W	Waits for completion of [axis] motion before next command is executed	
WHILE ENDWHILE	-	Continue executing the program after the WHILE statement when the conditional statement is true. Repeat until WHILE condition is false, and then execute program after ENDWHILE. Conditional operands available include: =, >, <, >=, <=, !=	2.7.2
X[position]	W	Individual move command for X-axis to [position]	2.2.4.2
Y[position]	W	Individual move command for Y-axis to [position]	

Command	R/W	Description	Reference Chapter
Z[position]	W	Individual move command for Z-axis to [position]	
U[position]	W	Individual move command for U-axis to [position]	
X[pos]Y[pos]Z[pos]U[pos]	W	Interpolation move command (include all axes that will be interpolated). EINT must be enabled (1).	2.2.4.2 2.3.1 2.3.4
ZMOVE[axis][+/-]	W	Perform the EZ Count operation. Moves [axis] the number of Z-index pulses specified by ZCNT in positive (+) or negative (-) direction before stopping.	2.2.5.1

Table 5-42 A-Script standalone commands

5.3 Error Codes

Error Code	Description	Incorrect Code Example
Unknown Error	Command cannot be recognized	PRGA
BIT Error	Specified bit does not exist	DO10=1
Token Error	Syntax error	DELAY=V1+200
= Error	Incorrect assignment formula	V2+1=V3
Assignment Error	Invalid value specified	V1= ABS
Operation Argument Error	Invalid command specified	V1=V2+ABS
Error not Number of Var	Invalid specification method	DO=PX
Var Range Error	Exceeds the defined variable range	V200=1
Missing =	Assignment operator '=' cannot be found	HSPD*1000
Error GOSUB	Specified value is outside the range for GOSUB	GOSUB900
Error SUB Number	Subroutine number is not valid	GOSUB DI1
Error SUB	Specified value is outside the range for SUB	SUB5000
Error ENDSUB	ENDSUB is missing	SUB 1 … ENDSUB
Error PRG	Specified value is outside the range for PRG	PRG200
Error PRG Number	Program number is not valid	PRG XI3
Missing END	END for PRG missing	PRG 0 … END
Missing ENDWHILE	ENDWHILE for WHILE missing	WHILE 1=1 … ENDWHILE

Missing ENDIF	ENDIF for IF missing	IF 1=1
		…
		ENDIF

Error	Other syntax errors	V1==V2

Table 5-43 A-Script error codes

5.4 Example Standalone Programs

The following examples are provided to assist your understanding. Although they were produced with the utmost care, if you find any points that are unclear, wrong, or have inadequate descriptions, please let us know.

For more examples please refer to the command refence.

5.4.1 Standalone Example Program 1 – Single Thread

Summary of operation

Set high and low speed and move the X-axis to 1000 and back to 0.

Program code

```
PRG 0                         ;Program start line, program 0
        HSPD=20000            ;Set high speed to 20,000 pulses/sec
        LSPD=1000             ;Set low speed to 1,000 pulses/sec
        ACC=300               ;Set acceleration to 300 msec
        EO=1                  ;Enable the X-axis
        X1000                 ;Move X-axis to 1,000
        WAITX                 ;Wait for X-axis move to complete
        X0                    ;Move X-axis to zero
        WAITX                 ;Wait for X-axis move to complete
END                           ;End of program 0
```

5.4.2 Standalone Example Program 2 – Single Thread with buffer

Summary of operation

Create a contouring profile for the X-axis using buffer move function.

Program code

```
PRG 1                                 ;Program start line, program 1
        BUFON                         ;Enable buffer move
; Load Buffer
        HSPD=100                      ;Set high speed to 100 pulses/sec
        X1000Y100Z100                 ;I:1000:100:100:100
        HSPD=200                      ;Set high speed to 200 pulses/sec
        X2000Y100Z100                 ;I:2000:100:100:200
        HSPD=300                      ;Set high speed to 300 pulses/sec
        X3000Y100Z100                 ;I:3000:100:100:300
        HSPD=400                      ;Set high speed to 400 pulses/sec
        X4000Y100Z100                 ;I:4000:100:100:400
        HSPD=500                      ;Set high speed to 500 pulses/sec
        X5000Y100Z100                 ;I:5000:100:100:500
        HSPD=400                      ;Set high speed to 400 pulses/sec
        X3000Y100Z100                 ;I:3000:100:100:400
        HSPD=300                      ;Set high speed to 300 pulses/sec
        X0Y100Z100                    ;I:0:100:100:300
; Start buffered move
        ISTART                        ;BSTART
        DELAY=100                     ;Delay 0.1 seconds
        WAITX                         ;Wait until axis X motion has stopped
BUFOFF                                ;Disable buffer move
END                                   ;End of program 1
```

5.4.3 Standalone Example Program 3 – Single Thread

Summary of operation

Move the X-axis back and forth indefinitely between position 1000 and 0.

Program code

```
PRG 1                           ;Program start line, program 1
        HSPD=20000              ;Set the high speed to 20000 pulses/sec
        LSPD=1000               ;Set the low speed to 1000 pulses/sec
        ACC=300                 ;Set the acceleration to 300 msec
        EO=1                    ;Enable the motor power
        WHILE 1=1               ;Forever loop
                X0              ;Move to zero
                WAITX           ;Wait for X-axis move to complete
                X1000           ;Move to 1000
                WAITX           ;Wait for X-axis move to complete
        ENDWHILE                ;Go back to WHILE statement

END                             ;End of program 1
```

5.4.4 Standalone Example Program 4 – Single Thread

Summary of operation

Move the X-axis back and forth 10 times between position 1000 and 0.

Program code

```
PRG 1                           ;Program start line, program 1
        HSPD=20000              ;Set high speed to 20,000 pulses/sec
        LSPD=1000               ;Set low speed to 1,000 pulses/sec
        ACC=300                 ;Set acceleration to 300 msec
        EO=1                    ;Enable the X-axis
        V1=0                    ;Set variable 1 to 0
        WHILE V1<10             ;Loop while variable 1 is less than 10
                X0              ;Move X-axis to zero (0)
                WAITX           ;Wait for X-axis move to complete
                X1000           ;Move X-axis to 1,000
                WAITX           ;Wait for X-axis move to complete
                V1=V1+1         ;Increment variable 1 by 1
        ENDWHILE                ;Go back to WHILE statement
END                             ;End of program 1
```

5.4.5 Standalone Example Program 5 – Single Thread

Summary of operation

Move the X-axis back and forth between position 1000 and 0 only if the digital input 1 is turned on.

Program code

```
PRG 2                                   ;Program start line, program 2
        HSPD=20000                      ;Set high speed to 20,000 pulses/sec
        LSPD=1000                       ;Set low speed to 1,000 pulses/sec
        ACC=300                         ;Set acceleration to 300 msec
        EO=1                            ;Enable the X-axis
        WHILE 1=1                       ;Forever loop
                IF DI1=1                ;If digital input 1 is on, execute the following commands
                        X0              ;Move X-axis to zero
                        WAITX           ;Wait for X-axis move to complete
                        X1000           ;Move X-axis to 1,000
                        WAITX           ;Wait for X-axis move to complete
                ENDIF                   ;End of the IF loop
        ENDWHILE                        ;Go back to WHILE Statement
END                                     ;End of program 2
```

5.4.6 Standalone Example Program 6 – Single Thread with subroutine

Summary of operation

Using a subroutine, increment the X-axis by 1000 whenever the DI1 rising edge is detected.

Program code

```
PRG 0                                   ;Program start line, Program 0
        HSPD=20000                      ;Set high speed to 20,000 pulses/sec
        LSPD=1000                       ;Set low speed to 1,000 pulses/sec
        ACC=300                         ;Set acceleration to 300 msec
        ABS                             ;Set move mode to absolute mode
        EO=1                            ;Enable the X-axis
        V1=0                            ;Set variable 1 to 0
        WHILE 1=1                       ;Forever loop
                IF DI1=1                ;If digital input 1 is on, execute the following commands
                        GOSUB 1         ;Execute subroutine 1
                ENDIF                   ;End of the IF loop
        ENDWHILE                        ;Go back to WHILE statement

END                                     ;End of program 0

SUB 1                                   ;Subroutine start line, subroutine 1
        XV1                             ;Move X-axis to V1 target position
        WAITX                           ;Wait for X-axis move to complete
        V1=V1+1000                      ;Increment V1 by 1,000
        WHILE DI1=1                     ;Wait until the DI1 is turned off (prevent multiple increments)
        ENDWHILE                        ;Go back to WHILE statement
ENDSUB                          ;End of subroutine 1
```

5.4.7 Standalone Example Program 7 – Single Thread with error handling

Summary of operation

If digital input 1 is on, move to position 2000. If digital input 2 is on, move to position 4000. If digital input 3 is on, move to 6000. If digital input 4 is on, home the motor in negative direction, mode 0. Use digital output 1 to indicate that the motor is moving or not moving. Also, create an error handling subroutine utilizing SUB 31 that clears the error and then returns the motor to position zero.

Program code

```
PRG 2                                   ;Start program 2
        HSPD=15000                      ;Set high speed to 15,000 pulses/sec
        LSPD=500                        ;Set low speed to 500 pulses/sec
        ACC=300                         ;Set acceleration to 300 msec
        EO=1                            ;Enable the X-axis
        WHILE 1=1                       ;Forever loop
                IF DI1=1                ;If digital input 1 is on
                        X2000           ;Move X-axis to 1000
                ELSEIF DI2=1            ;If digital input 2 is on
                        X4000           ;Move X-axis to 4000
                ELSEIF DI3=1            ;If digital input 3 is on
                        X6000           ;Move X-axis to 6000
                ELSEIF DI4=1            ;If digital input 4 is on
                        HOMEX-0         ;Home X-axis with mode 0 in the negative direction
                ENDIF                   ;End of the IF loop
                V1=MSTX                 ;Store the X-axis status to variable 1
                V2=V1&7                 ;Get first 3 bits
                IF V2!=0                ;If one of first 3 bits is high (X-axis moving)
                        DO1=1           ;Turn on digital output 1
                ELSE                    ;Else if first 3 bits are low (X-axis idle)
                        DO1=0           ;Turn off digital output 1
                ENDIF                   ;End of the IF loop
        ENDWHILE                        ;Go back to WHILE statement

END                                     ;End of program 2

SUB 31                                  ;Subroutine start line, subroutine 31 (error handling subroutine)
        ECLEARX                         ;Clear the current error in X-axis
        X0                              ;Return X-axis to position 0
        DO2=1                           ;Turn on digital output 2
ENDSUB                                  ;End of subroutine 31
```

5.4.8 Standalone Example Program 8 – Multi-thread

Summary of operation

Program 0 will continuously move the motor between positions 0 and 1000. Simultaneously, program 1 will control the status of program 0 using digital inputs.

Program code

```
PRG 0                                   ;Start program 0
        HSPD=20000                      ;Set speed to 20,000 pulses/sec
        LSPD=500                        ;Set speed to 500 pulses/sec
        ACC=500                         ;Set acceleration to 500 msec
        WHILE 1=1                       ;Forever loop

                X0                      ;Move X-axis to position 0
                WAITX                   ;Wait for X-axis move to complete
                X1000                   ;Move X-axis to position 1,000
                WAITX                   ;Wait for X-axis move to complete

        ENDWHILE                        ;Go back to WHILE statement

END                                     ;End program 0

PRG 1                                   ;Start program 1
        WHILE 1=1                       ;Forever loop
                IF DI1=1                ;If digital input 1 is triggered
                        ABORTX          ;Stop movement in X-axis
                        SR0=0           ;Stop program 0
                ELSE                    ;If digital input 1 is not triggered
                        SR0=1           ;Run program 0
                ENDIF                   ;End of the IF loop
        ENDWHILE                        ;Go back to WHILE statement

END                                     ;End program 1
```

5.4.9 Standalone Example Program 9 – Multi-thread with subroutine

Summary of operation

Program 0 will continuously move the X-axis between positions 0 and 1000. Simultaneously, program 1 will monitor the communication time-out parameter and trigger digital output 1 if a time-out occurs. Program 1 will also stop all motion, disable program 0 and then re-enable it after a delay of 3 seconds when an error occurs.

Program code

```
PRG 0                                   ;Start program 0
        HSPD=1000                       ;Set high speed to 1,000 pulses/sec
        LSPD=500                        ;Set low speed to 500 pulses/sec
        ACC=500                         ;Set acceleration to 500 msec
        EO=1                            ;Enable X-axis
        GOSUB 1                         ;Go to Subroutine 1
        WHILE 1=1                       ;Forever loop
                X0                      ;Move X-axis to position 0
                WAITX                   ;Wait for X-axis move to complete
                X1000                   ;Move X-axis to position 1000
                WAITX                   ;Wait for X-axis move to complete
        ENDWHILE                        ;Go back to WHILE statement
END                                     ;End program 0

PRG 1                                   ;Start program 1
        WHILE 1=1                       ;Forever loop
                V1=MSTX&1024            ;Get bit alarm error variable
                IF V1 = 1024            ;If alarm error is on
                        SR0=0           ;Stop program 0
                        ABORTX          ;Abort X-axis motion
                        DO=0            ;Set digital output 0 to 0 (off)
                        DELAY=3000      ;Delay 3 seconds
                        ECLEARX         ;Clear alarm error
                        SR0=1           ;Turn program 0 on
                        DO=1            ;Set digital output 0 to 1 (on)
                ENDIF                   ;End of the IF loop
        ENDWHILE                        ;Go back to WHILE statement
END                                     ;End program

SUB 1                                   ;Subroutine start line, subroutine 1
        HOMEX-6                         ;Home X-axis with mode 6 in the negative direction
        WAITX                           ;Wait for X-axis move to complete
ENDSUB                                  ;End of subroutine 1

SUB 31                                  ;Subroutine start line, subroutine 31 (error handling subroutine)
        ECLEARX                         ;Clear the current error in X-axis
        X0                              ;Return X-axis to position 0
        DO2=1                           ;Turn on digital output 2
ENDSUB                                  ;End of subroutine 31
```

5.4.10 Standalone Example Program 10 – Multi-thread with subroutine

Summary of operation

Program 1 will home axis X and Y, then will continuously move the X and Y-axis and control a laser pointer to draw circle, line, then an Arc. Simultaneously, program 3 will control the status of general-purpose digital outputs to control LED's. Subroutine 1 turns off a laser pointer, Subroutine 2 turns on the laser pointer, Subroutine 7 enables and homes the X and Y-axis, and Subroutine 31 handles errors.

Program code

```
PRG 1                                         ;Start program 1
        GOSUB 1                               ;Go to subroutine 1, Turn off laser pointer
        GOSUB 7                               ;Go to subroutine 7,  Enable and Home X/Y
        WHILE 1=1                             ;Forever loop
; Draw Circle
                X85610Y562530         ;Move to start point for Circle
                WAITX                         ;Wait for X-axis move to complete
                GOSUB 2                       ;Go to subroutine 2, Turn on laser pointer
                CIRXYP487659:562431           ;Moves XY in a CW circle around points X487659 Y562431
                WAITX                         ;Wait for X-axis move to complete
                GOSUB 1                       ;Go to subroutine 1, Turn off laser pointer
; Draw Line
                X100000Y100000                ;Move to start point of Line
                WAITX                         ;Wait for X-axis move to complete
                GOSUB 2                       ;Go to subroutine 2, Turn on laser pointer
                X909000Y642390        ;Linear XY move from X100000 Y100000 to X909000 Y642390
                WAITX                         ;Wait for X-axis move to complete
                GOSUB 1                       ;Go to subroutine 1, Turn off laser pointer
; Draw Arc
                X430589Y20623         ;Move to start point for Arc
                WAITX                         ;Wait for X-axis move to complete
                GOSUB 2                       ;Go to subroutine 2, Turn on laser pointer
                ARCXYN288899:267403:95000     ;270dig arc of XY CCW around points X288899 Y267403
                WAITY                         ;Wait for Y-axis move to complete
                GOSUB 1                       ;Go to subroutine 1, Turn off laser pointer
        ENDWHILE                              ;Go back to WHILE statement
END                                           ;End program 1

PRG 3                                         ;Start program 3
        WHILE 1 = 1                           ;Forever loop
                IO=64                         ;General-purpose output 7 on.
                DELAY=700                     ;Delay 0.7 seconds
                IO=80                         ;General-purpose output 5 and 7 on.
                DELAY=700                     ;Delay 0.7 seconds
                IO=84                         ;General-purpose output 3, 5, and 7 on.
                DELAY=700                     ;Delay 0.7 seconds
                IO=85                         ;General-purpose output 1, 3, 5, and 7 on.
                DELAY=700                     ;Delay 0.7 seconds
                V30=0                         ; Set variable 30 to 0.
                WHILE V30 < 10                ;Loop while variable 30 is less than 10
                        IO = 85               ;General-purpose output 1, 3, 5, and 7 on.
                        DELAY=50              ;Delay 0.05 seconds
                        IO = 0                ;All general-purpose outputs off.
                        DELAY=50              ;Delay 0.05 seconds
                        V30=V30+1               ;Increment variable 30 by 1
                ENDWHILE                        ;Go back to WHILE statement
        ENDWHILE                                ;Go back to WHILE statement
END                                             ;End Program 3

SUB 1                                           ;Subroutine start line, subroutine 1
        DO2=0                                   ;DO 2 off to turn off Laser pointer
        DELAY=10                                ;Delay 0.01 seconds to stabilize laser
ENDSUB                                  ;End of subroutine 1

SUB 2                                           ;Subroutine start line, subroutine 2
        DO2=1                                   ;DO 2 on to turn on Laser pointer
        DELAY=10                                ;Delay 0.01 seconds to stabilize laser
ENDSUB                                  ;End of subroutine 2

SUB 7                                           ;Subroutine start line, subroutine 7
; Enable motors
        HSPD=15000                              ;Set high speed to 15,000 pulses/sec
        LSPD=500                                ;Set low speed to 500 pulses/sec
        ACC=100                                 ;Set acceleration to 100 msec
        EO=3                                    ;Enable X and Y-axis
; Home axis
        HOMEX-6                                 ;Home X-axis with mode 6 in the negative direction
        WAITX                                   ;Wait for X-axis move to complete
        HOMEY-6                                 ;Home Y-axis with mode 6 in the negative direction
        WAITY                                   ;Wait for Y-axis move to complete
; Move X/Y to Start Position
        HSPD=35000                              ;Set high speed to 15,000 pulses/sec
        X500000Y500000                  ;Move X and Y-axis to X500,000 Y500,000
        WAITX                                   ;Wait for X-axis move to complete
ENDSUB                                  ;End of subroutine 7

SUB 31                                          ;Subroutine start line, subroutine 31 (error handling)
        SR1=0                                   ;Stop program 1
        SR3=0                                   ;Stop program 3
        ABORT                                   ;Abort all-axis motion
        ECLEARX                         ;Clear the current error in X-axis
        ECLEARY                         ;Clear the current error in Y-axis
        IO=0                                    ;All general-purpose outputs off.
        DO2=0                                   ;DO 2 off to turn off Laser pointer
ENDSUB                                  ;End of subroutine 7
```

6.0 Dimensions and Connectivity

6.1 Dimensions

All dimensions in mm. Tolerance ±0.1

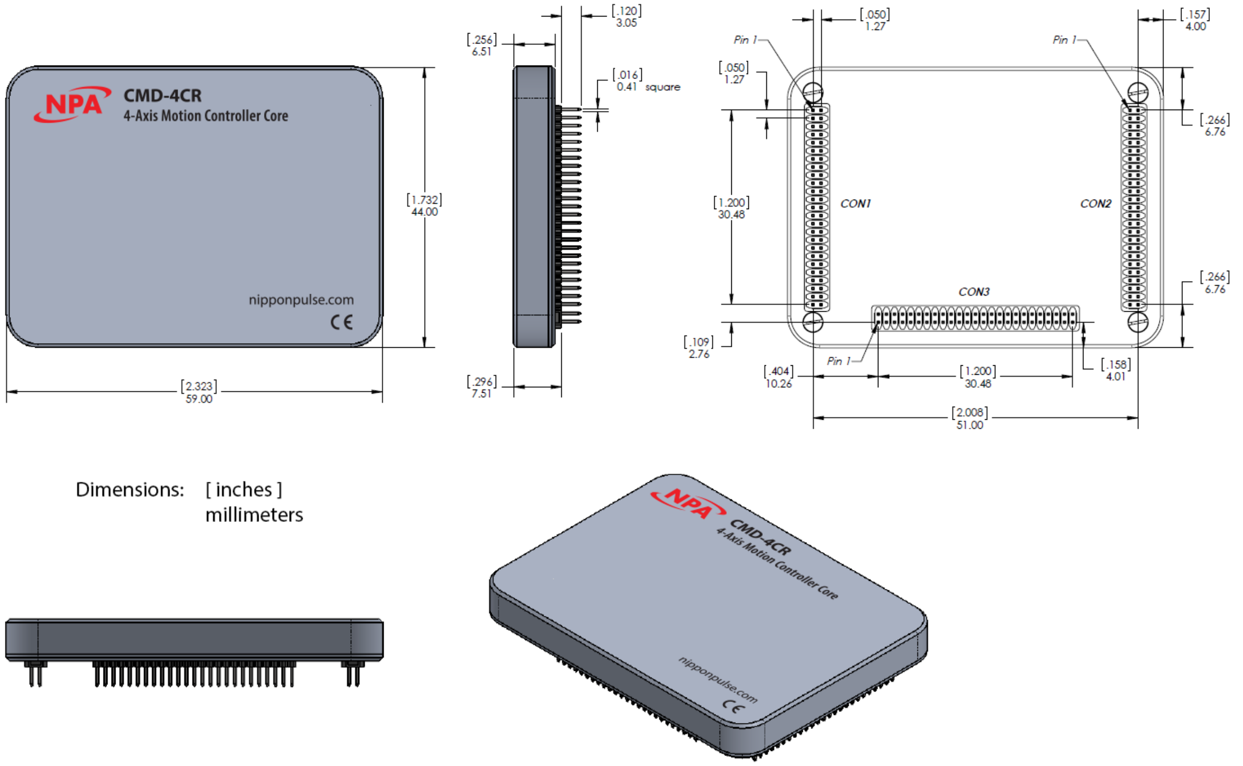


Figure 6-41

6.2 Connectivity

The Commander core uses 3 x 50-pin connectors with 1.27mm pitch to allow access to all signals. In this manual – when viewing from the bottom, with the three sets of pins, left, right and bottom – we will refer to the left set of pins as CON1, the right set as CON2, and the bottom set as CON3. For the purpose of simplicity, in this manual, all pin numbers are referred to by CON number then pin number in the set. For example, pin 3 on CON1 will be referred to as pin 1-3, whereas pin 25 on CON3 will be referred to as pin 3-25.

Recommended Mating Connector Information

Manufacturer: FCI

Part Number: 20021311-00050T4LF

Other compatible 1.27mm connectors can be used. Figure 6-42 shows the Pin 1 locations.

6.2.1 Pinout

As viewed from the bottom looking at the unit.

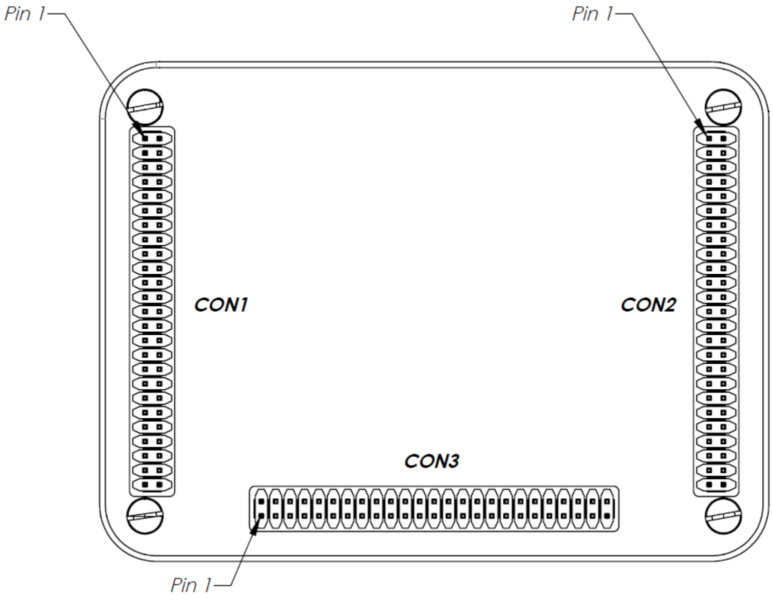


Figure 6-42

6.1.1.1 Pin Assignment Diagram

As viewed from the top looking through the device. (As laid out on PCB)

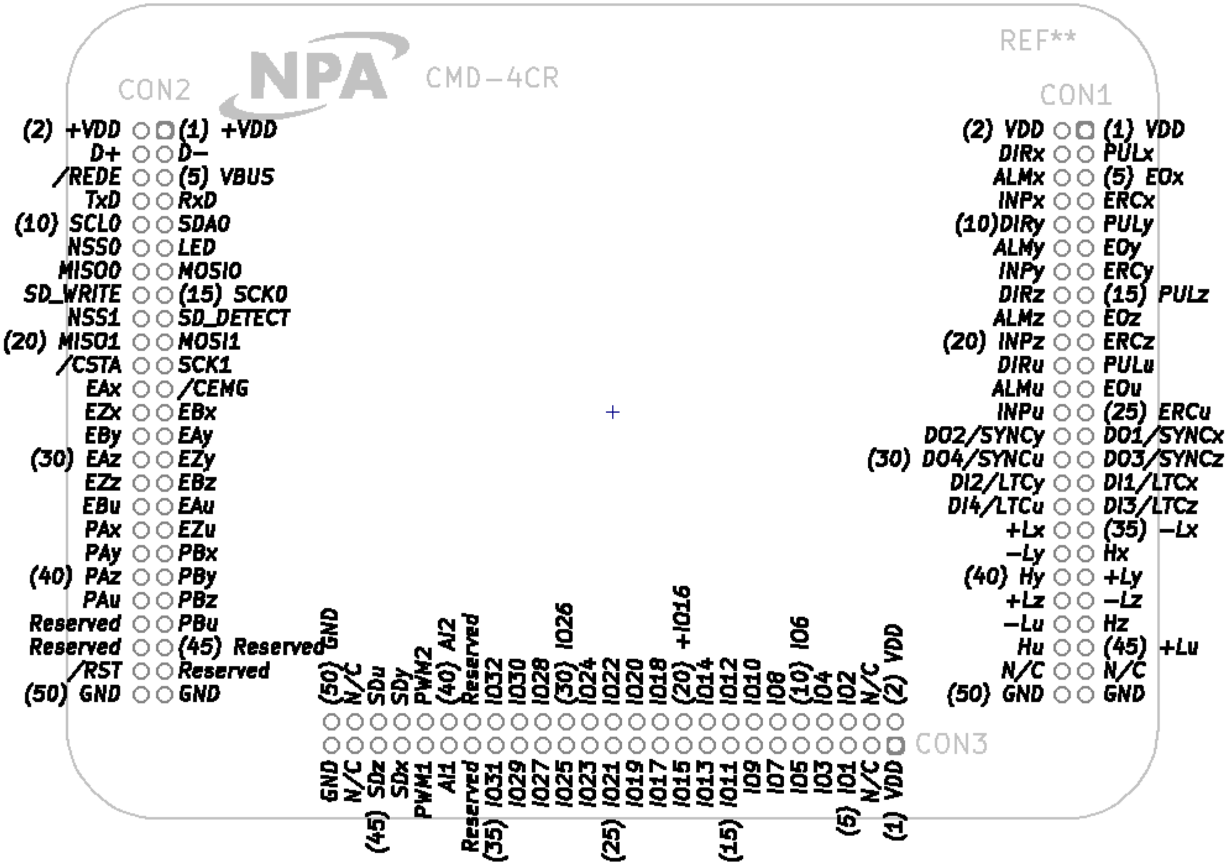


Figure 6-43

6.1.1.2 Pin Descriptions by Functions

Function	Signal Name	Terminal Number	Input/Output	Logic	Description
Power					
Ground	GND	1-49, 1-50, 2-49, 2-50, 3-49, 3-50	Power Source		Common connection for power supply. Make sure to connect all terminals.
Voltage	VDD	1-1, 1-2, 2-1, 2-2, 3-1, 3-2	Power Source		Supply +3.3 VDC power. The allowable power supply range is +3.3 VDC ±10%. Make sure to connect all terminals.
Reset	/RST	2-48	Input	Negative	Input reset signal. An external switch can be installed for soft reset of the Commander. To perform a soft reset, hold /RST LOW for at least 0.5ms.
Power Indicator	LED	2-11	Output	Negative	LED power indicator. Also used to show firmware bootloading status.
Motion					
Pulse	PULx PULy PULz PULu	1-3 1-9 1-15 1-21	Output	Negative#	Output command pulses for controlling a motor. **When common pulse mode is selected:** Outputs pulses and the feed direction is determined by DIR signals. **When two-pulse output mode is selected:** Outputs pulses in the positive (+) direction. **When 90 phase difference mode is selected:** Outputs DIR signals and 90 phase difference signals. The output logic can be changed using software.

Function	Signal Name	Terminal Number	Input/Output	Logic	Description
Direction	DIRx DIRy DIRz DIRu	1-4 1-10 1-16 1-22	Output	Negative#	Output command pulses for controlling a motor, or outputs direction signal. **When common pulse mode is selected:** Outputs a direction signal. **When two-pulse output mode is selected:** Outputs pulses in the negative (-) direction. **When 90 phase difference mode is selected:** Outputs DIR signals and 90 phase difference signals. The output logic can be changed using software
Enable	EOx EOy EOz EOu	1-5 1-11 1-17 1-23	Output	Negative#	Outputs a motor enable signal to the driver. Output logic can be changed using software.
In-Position	INPx INPy INPz INPu	1-8 1-14 1-20 1-28	Input U	Negative#	Input the position complete signal from servo driver (in-position signal). Input logic can be changed using software.
Servo Alarm	ALMx ALMy ALMz ALMu	1-6 1-12 1-18 1-24	Input U	Negative#	Input servo alarm signal. When this signal is ON, motion of an axis stops immediately, or will decelerate and stop. The input logic can be changed using software.
Error Clear	ERCx ERCy ERCz ERCu	1-7 1-13 1-19 1-25	Output	Negative#	Outputs a servo error clear (deflection counter clear) signal to a servo driver as a pulse. The output logic and pulse width can be changed using software. A LEVEL signal output is also available.
Safety					
Simultaneous Start	/CSTA	2-22	Input/Output	Negative	Input/output terminal for simultaneous start. See 2.1.1.2 Synchronization with external trigger
E-Stop	/CEMG	2-23	Input U	Negative	Input for an emergency stop. While this signal is LOW, motion cannot start. If this signal changes to LOW while in operation, all the motors will stop operation immediately.

Function	Signal Name	Terminal Number	Input/Output	Logic	Description
Limits	+Lx +Ly +Lz +Lu	1-36 1-39 1-42 1-45	Input U	Negative#	Input end limit signal in the positive (+) direction. When this signal is ON while feeding in the positive (+) direction, motion of an axis will stop immediately or will decelerate and stop. The input logic can be selected using software.
	-Lx -Ly -Lz -Lu	1-35 1-38 1-41 1-44	Input U	Negative#	Input end limit signal in the negative (-) direction. When this signal is ON while feeding in negative (-) direction, motion of an axis will stop immediately, or will decelerate and stop. The input logic can be selected using software.
Slow Down	SDx SDy SDz SDu	3-43 3-44 3-45 3-46	Input U	Negative#	Input for slowdown (deceleration stop) signal. Selects the input method: LEVEL or LATCHED inputs. The input logic can be selected using software.
Home	Hx Hy Hz Hu	1-37 1-40 1-43 1-46	Input U	Negative#	Input home (origin position) signal. Used for origin position operations (edge detection). The input logic can be selected using software.
Feedback					
Encoder	EAx, EBx EAy, EBy EAz, EBz EAu, EBu	2-24, 2-24 2-27, 2-28 2-30, 2-31 2-33, 2-34	Input U		Input this signal when you want to control the mechanical position using the encoder signal. Input a 90 phase difference signal (1x, 2x, 4x) or input positive (+) pulses on EA and negative (-) pulses on EB. (Two-pulse input) When inputting 90 phase difference signals, if the EA signal phase is ahead of the EB signal, the motion core will count up (count forward) pulses. The counting direction can be changed using software.
Index	EZx EZy EZz EZy	2-26 2-29 2-32 2-35	Input U	Negative#	Input a index (marker) signal when using the marker signal in origin return mode. (This signal is output once for each turn of the encoder.) Use of the EZ signal improves origin return precision. The input logic can be changed using software

Function	Signal Name	Terminal Number	Input/Output	Logic	Description
Input/Output					
Digital I/O	DI1/LTCx DI2/LTCy DI3/LTCz DI4/LTCu	1-31 1-32 1-33 1-34	Input U	Negative#	High-speed input. When Latch function is enabled, will lock encoder and pulse out count. The input logic can be changed using software.
	D01/SYNCx DO2/SYNCy DO3/SYNCz DO4/SYNCu	1-27 1-28 1-29 1-30	Output	Positive	High-speed output. When synchronization function is enabled, will output synchronization based on synchronization mode. The input logic can be changed using software.
	IO1 to IO32	3-5 to 3-36	Input/Output		General-purpose, configurable I/O
Analog	AI1 AI2	3-39 3-40	Input		Analog input 10-bit 0-3.3V
	PWM1 PWM2	3-41 3-42	Output		PWM output
MPG	PAx, PBx PAy, PBy PAz, PBz PAu, PBu	2-36, 2-37 2-38, 2-29 2-40, 2-41 2-42, 2-43	Input U		Input for receiving external drive pulses, such as manual pulse generator. Input 90 phase difference signals (1x, 2x, 4x) or positive (+) pulses (on PA) and negative (-) pulses (on PB). (Two-pulse input) When 90 phase difference signals are used, if the signal phase of PA is ahead of the PB signal, the Commander will count up (count forward) pulses. The counting direction can be changed using software.
Communication					
USB	D+ D- VBUS	2-4 2-3 2-5	Input/Output Input/Output Input		USB D+ USB D- USB sense input
RS485	/REDE RxD TxD	2-6 2-7 2-8	Output Input Output		Receive/transmit signal for serial communication RXD signal for serial communication TXD signal for serial communication
I^2C	SDA0 SCL0	2-9 2-10	Input/Output Output		SDA [I^2C Channel 0] SCL [I^2C Channel 0]

Function	Signal Name	Terminal Number	Input/Output	Logic	Description
SPI	NSS0 MOSI0 MISO0 SCK0	2-12 2-12 2-14 2-15			Secondary select [SPI Channel 0] Main out, secondary in [SP! Channel 0] Main in, secondary out [SP! Channel 0] Serial clock [SP! Channel 0]
	NSS1 MOSI1 MISO1 SCK1	2-18 2-19 2-20 2-21			Secondary select [SPI Channel 1] Main out, secondary in [SP! Channel 1] Main in, secondary out [SP! Channel 1] Serial clock [SP! Channel 1]
SD	SD_Write SD_Detect	2-16 2-17			These pins are reserved for future firmware updates, please do not connect these pins.
No Connection					
N/C		1-47, 1-48 3-3, 3-4 3-47, 3-48	N/C		These pins have no connections
Reserved		2-44 to 2-47, 3-37, 3-38	N/C		These pins are reserved for future firmware updates, please do not connect these pins.

Note 1: "Input U" refers to an input with a pull up resistor. The internal pull up resistance (40 k to 240 k-ohms) is only used to keep a terminal from floating. If you want to use the Commander core with an open collector system, an external pull up resistor (5k to 10 k-ohms) is required.
As a noise prevention measure, pull up unused terminals to VDD using an external resistor (5 k to 10k-ohms), or connect them directly to VDD5.

Note 2: "Input/Output *" refers to a terminal with a pull up resistor. The internal pull up resistor (40 k to 240 k-ohms) is only used to keep a terminal from floating. If it is connected in a wired OR circuit, an external pull up resistor (5 k to 10 k-ohms) is required.
As a noise prevention measure, pull up unused terminals to VDD using an external resistor (5 k to 10 k-ohms).

Note 3: If an output terminal is not being used, leave it open.

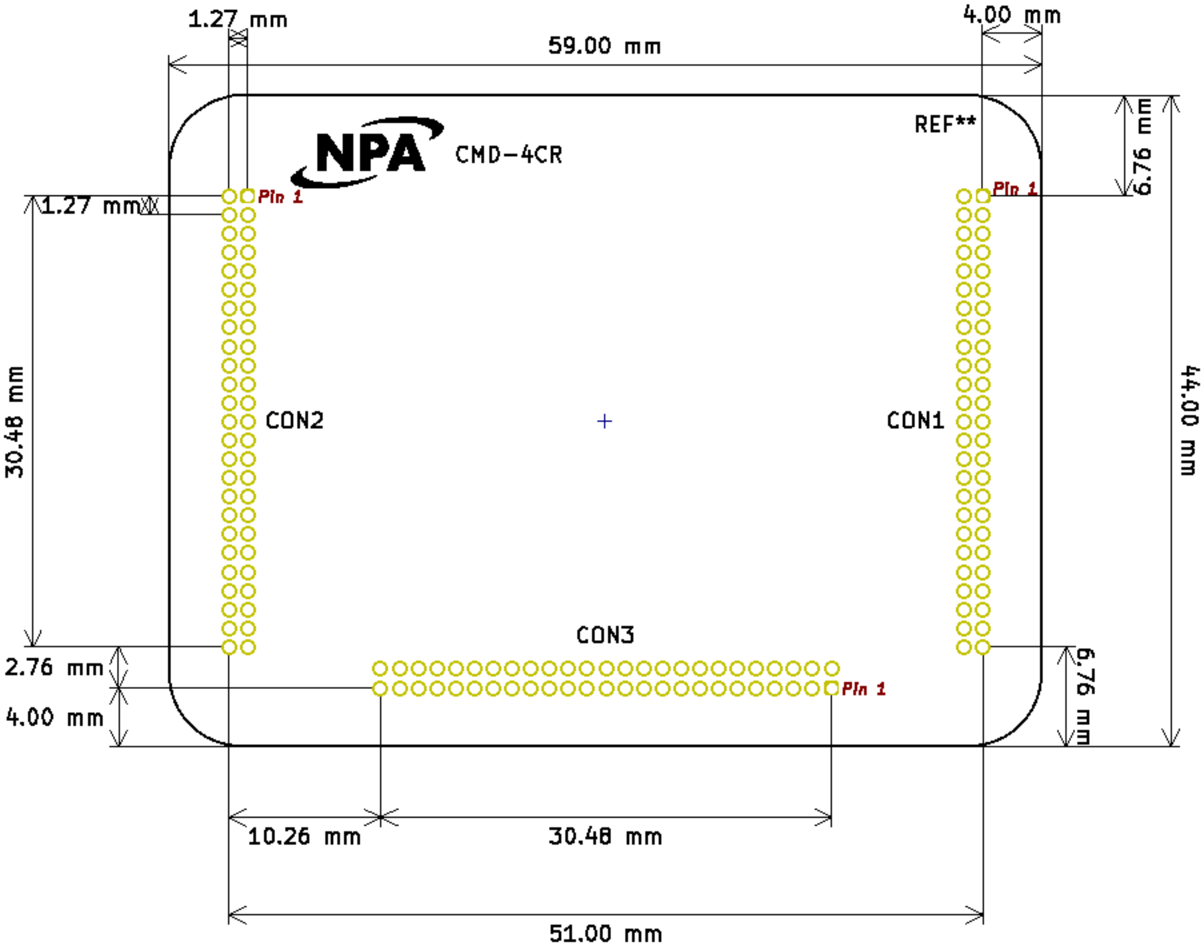
Note 4: "Positive" refers to positive logic. "Negative" refers to negative logic. "#" means that the logic can be changed using software. The logic shown refers only to the initial status of the terminal.

Note 5: Use the POL[axis] command to select an output signal.

Note 6: When a deceleration stop (Slow Down) is selected, keep the input signal ON until an axis stops.

Note 7: ORG input is synchronized with output pulses, sampled and controlled by a change of sampling result. Therefore, keep ORG sensor ON for longer than feed amount for one pulse.

6.1.1.3 Recommended Footprint



6.2.2 Interface Boards

Nippon Pulse provides off-the-shelf interface board designs specifically for the Commander core. These provide quick and easy access to all the features and technology of the Commander core.

6.1.2.1 Development Kit

A development kit that includes a Commander core, an interface PCB, related cables, and software (via download) is available. Every available feature and signal on the Commander core will be available through the development kit (CMD-4CR-EV), which can function as the launchpad for your custom motion solution. Contact Nippon Pulse for more information on the development kit.

7.0 Electrical Characteristics

7.1 Electrical and Thermal Specifications

Parameter	Min	Typ	Max	Units
Power				
Main Power Input (V_{DD})	+3	+3.3	+3.6	V
			500	mA
VBUS (USB Sense)	0	+5	+5.8	V
I/O				
Digital Outputs	0		V_{DD}	V
	-6		6	mA
Digital Output High Voltage	V_{DD} - 0.6		V_{DD}	V
Digital Output Low Voltage	0	0.4	0.6	V
Digital Inputs	-0.3		V_{DD}	V
	-90		30	uA
Digital Input High Voltage	V_{DD} - 0.6		5.8	V
Digital Input Low Voltage	0		0.6	V
Analog Inputs	0		V_{DD}	V
			1.5	mA
Analog Outputs	0		V_{DD}	V
			2.0	mA
Thermal				
Operating Temperature	-40		+85	°C
Storage Temperature	-55		+150	°C
Flash Memory				
Endurance (Write/Erase Cycles)	20K	100K		Cycles
Retention (25°C, 1K Cycles)	10	100		Years

Table 7-44

7.2 Handling Precautions

7.2.1 Design precautions

1) Never exceed the absolute maximum ratings, even for a very short time.
2) Take precautions against the influence of heat in the environment keeping the temperature around the Commander core as cool as possible.
3) Please note that ignoring the following may result in latching up and may cause overheating and smoke.
   a. Do not apply a voltage greater than the absolute maximum rating voltage. Please consider the voltage drop timing when turning the power ON/OFF.
   b. Be careful not to introduce external noise into the Commander core.
   c. Hold the unused input terminals to +3.3V or GND level.
   d. Do not short-circuit the outputs.
   e. Protect the LSI from inductive pulses caused by electrical sources that generate large voltage surges, and take appropriate precautions against static electricity.
4) Provide external circuit protection components so that overvoltage caused by noise, voltage surges, or static electricity are not fed to the Commander core.

7.2.2 Precautions for transporting and storing Commander cores

1) Always handle carefully and keep in their packages. Throwing or dropping may damage them.
2) Do not store in a location exposed to water droplets or direct sunlight.
3) Do not store in a location where corrosive gases are present, or in excessively dusty environments.
4) Store in an anti-static storage container, and make sure that no physical load is placed on the Commander cores.

7.2.3 Precautions for installation

1) In order to prevent damage caused by static electricity, pay attention to the following.
   a. Make sure to ground all equipment, tools, and jigs that are present at the work site.
   b. Ground the work desk surface using a conductive mat or similar apparatus (with an appropriate resistance factor). However, do not allow work on a metal surface, which can cause a rapid change in the electrical charge due to extremely low resistance.
   c. When using a vacuum device, provide anti-static protection using a conductive rubber pickup tip. Anything which contacts the leads should have as high a resistance as possible.
   d. When using a pincer that may make contact with any terminals, use an anti-static model. Do not use a metal pincer, if possible.
   e. Store unused commander cores in a PC board storage box that is protected against static electricity, and make sure there is adequate clearance between the Commander cores. Never directly stack them on each other, as it may cause friction that can develop an electrical charge.
2) Operators must wear wrist straps which are grounded through approximately 1M-ohm of resistance.
3) Use low voltage soldering devices and make sure the tips are grounded.
4) Do not store or use Commander cores near high-voltage electrical fields, such those produced by a CRT.
5) Please avoid soldering in a soaking method not so as to give a dramatic change of temperature to a package and change and not so as to damage to a device.

7.2.4 Other precautions

1) The package is made of fire-retardant material; however, it can burn. When baked or burned, it may generate gases or fire. Do not use it near ignition sources or flammable objects.
2) This Commander core is designed for use in commercial apparatus (office machines, communication equipment, measuring equipment, and household appliances). If you use it in any device that may require high quality and reliability, or where faults or malfunctions may directly affect human survival or injure humans, such as in nuclear power control devices, aviation devices or spacecraft, traffic signals, fire control, or various types of safety devices, we will not be liable for any problem that occurs, even if it was directly caused by the Commander core. Customers must provide their own safety measures to ensure appropriate performance in all circumstances.

Appendix A: Homing Mode Actions

Commander core utilizes 13 homing search methods, utilizing home limit, end of travel limit, or Z-index pulse, or combinations of these to complete a homing operation. Homing is used to establish a zero, or home, position to use as a reference for absolute motion actions. Some homing methods will stop in a non-zero position, using the position at which the home, end of travel, or Z-index pulse was triggered as the zero-reference position.

The chart below indicates which type of limits are used in each mode, along with home approach speed and direction and end-point status. ● indicates input required, ○ indicates input optional.

Homing Mode	Home	End Limits	Z-index	Slow Down	Approaches Home at		Direction to Trigger Zero	Ending Point
					High Speed	Low Speed		
Mode 0	●			○	**H**		Dir of Home	Non-Zero
Mode 1	●			○		**L**	Dir of Home	Zero
Mode 2	●		●	○		**L**	Dir of Home	Zero
Mode 3	●		●	○	**H**		Dir of Home	Non-Zero
Mode 4	●		●	○		**L**	Reverse Dir	Zero
Mode 5	●		●	○	**H**		Reverse Dir	Non-Zero
Mode 6		●		○	**H**		Reverse Dir	Zero
Mode 7		●	●	○		**L**	Reverse Dir	Zero
Mode 8		●	●	○	**H**		Reverse Dir	Non-Zero
Mode 9	●			○	**H**		Reverse Dir	Zero
Mode 10	●		●	○	**H**		Reverse Dir	Zero
Mode 11	●		●	○	**H**		Dir of Home	Zero
Mode 12		●	●	○	**H**		Dir of Home	Zero

Table A-45 Homing inputs

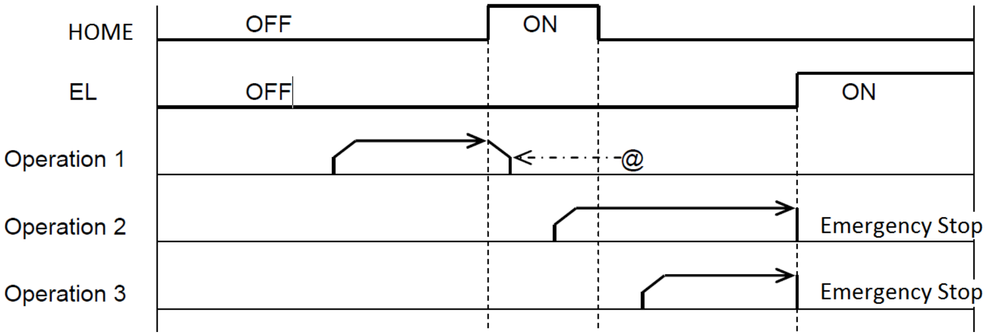
Consider the system mechanics carefully when setting up limit switches, using precaution to set the limits for the appropriate axis movement and consider the deceleration distance when establishing end of travel limits relative to the hard stop positions.

NOTE: For homing modes 9 – 12, encoder feedback is required. When using a stepper motor and homing with modes 9 – 12, an encoder is required and the command pulse count and encoder pulse count must be equal.

A.1 Homing Mode 0

Homing mode 0 with Slow Down (SD) disabled. This homing method works with a home limit switch input and sets home when the motion stops after the home limit switch is encountered.

Homing Mode	Home Limit	End Limits	Z-index	Slow Down	Approaches Home at		Direction to Trigger Zero	Ending Point
					High Speed	Low Speed		
Mode 0	●			○	**H**		Dir of Home	Non-Zero



Operation 1:

A) Homing is started toward home switch, accelerate from LSPD to HSPD.
B) Deceleration starts when the home (ORG) input point is detected, zero position is set.
C) Homing is complete when deceleration stops.
D) ERC pulse is sent.

Operation 2:

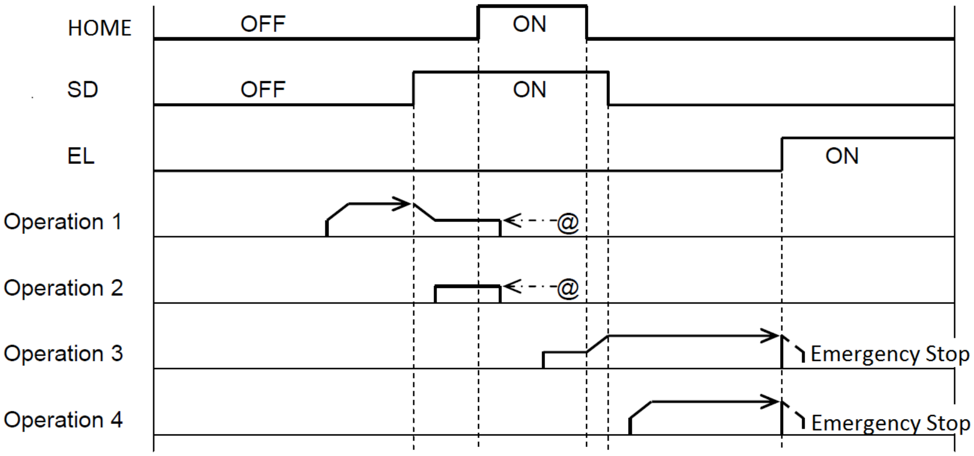
A) Homing is started while home input is activated.
B) Acceleration from LSPD to HSPD.
C) End limit is encountered, immediate stop.

Operation 3:

A) Homing is started moving away from the home input switch.
B) Acceleration from LSPD to HSPD.
C) End limit is encountered, immediate stop.

Homing mode 0 with Slow Down (SD) enabled. In this homing mode, an input will first trigger a deceleration to a lower speed point and then stops and sets home as the home limit switch is reached.

Homing Mode	Home Limit	End Limits	Z-index	Slow Down	Approaches Home at		Direction to Trigger Zero	Ending Point
					High Speed	Low Speed		
Mode 0	●			●		L	Dir of Home	Non-Zero



Operation 1:

A) Homing is started toward home switch, accelerate from LSPD to HSPD.
B) Deceleration starts to LSPD when the SD input point is detected, continue towards home.
C) Stop when home input is detected, homing complete.
D) ERC pulse is sent.

Operation 2:

A) Homing is started while SD input is activated.
B) Continue traveling at LSPD towards home input.
C) Stop when home input is detected, zero position is set, homing complete.
D) ERC pulse is sent.

Operation 3:

D) Homing is started while SD input is activated and is moving away from the home input switch.
E) Travel at LSPD until SD input is deactivated, then accelerate from LSPD to HSPD.
F) End limit is encountered, immediate stop.

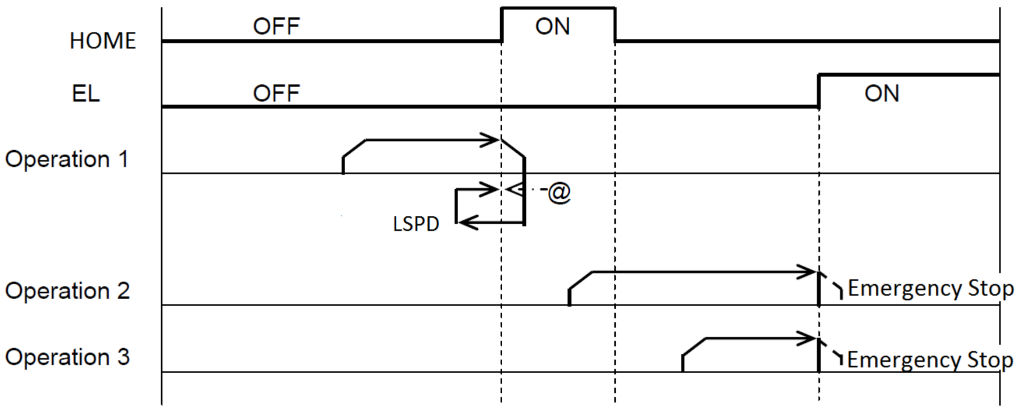
Operation 4:

A) Homing is started moving away from the home input switch and SD limit is deactivated.
B) Acceleration from LSPD to HSPD.
C) End limit is encountered, immediate stop.

A.2 Homing Mode 1

This homing method decelerates when it first encounters the home limit switch, reverses direction to back off the limit switch, and then approaches the home limit switch again at a lower speed point and sets home when the home limit switch is encountered a second time.

Homing Mode	Home Limit	End Limits	Z-index	Slow Down	Approaches Home at		Direction to Trigger Zero	Ending Point
					High Speed	Low Speed		
Mode 1	●			○		L	Dir of Home	Zero



Operation 1:

A) Homing operation is started, accelerate from LSPD to HSPD.
B) Deceleration starts when home limit is encountered.
C) Reverse direction at move at LSPD towards home limit.
D) Move at LSPD until home limit is turned off and reverse direction.
E) Operation stops when home limit is encountered again, zero position is set.
F) ERC pulse is sent.

Operation 2:

A) Homing is started while home input is activated.
B) Acceleration from LSPD to HSPD.
C) End limit is encountered, immediate stop.

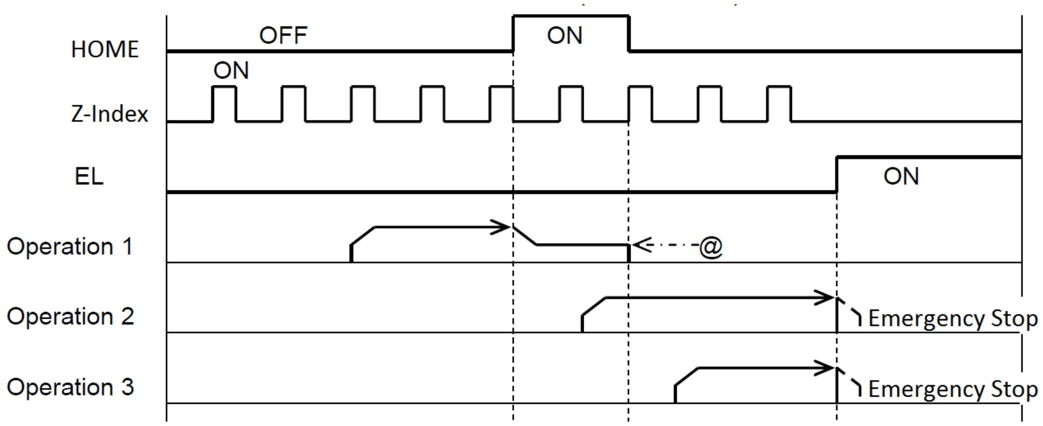
Operation 3:

A) Homing is started moving away from the home input switch.
B) Acceleration from LSPD to HSPD.
C) End limit is encountered, immediate stop.

A.3 Homing Mode 2

Homing method 2 utilizes both a home limit switch and the Z-index pulse of the encoder. Once the home limit switch is encountered, the motor slows to a lower speed point and sets home as it encounters the Z-index pulse.

Homing Mode	Home Limit	End Limits	Z-index	Slow Down	Approaches Home at		Direction to Trigger Zero	Ending Point
					High Speed	Low Speed		
Mode 2	●		●	○		L	Dir of Home	Zero



Operation 1:

A) Homing operation is started, accelerate from LSPD to HSPD.
B) Deceleration starts when home limit is encountered.
C) Continue moving at LSPD until specified number of Z-Index pulses are counted (ZCNT), zero position is set.
D) Operation stops and is completed.
E) ERC pulse is sent.

Operation 2:

A) Homing is started while home input is activated.
B) Acceleration from LSPD to HSPD.
C) End limit is encountered, immediate stop.

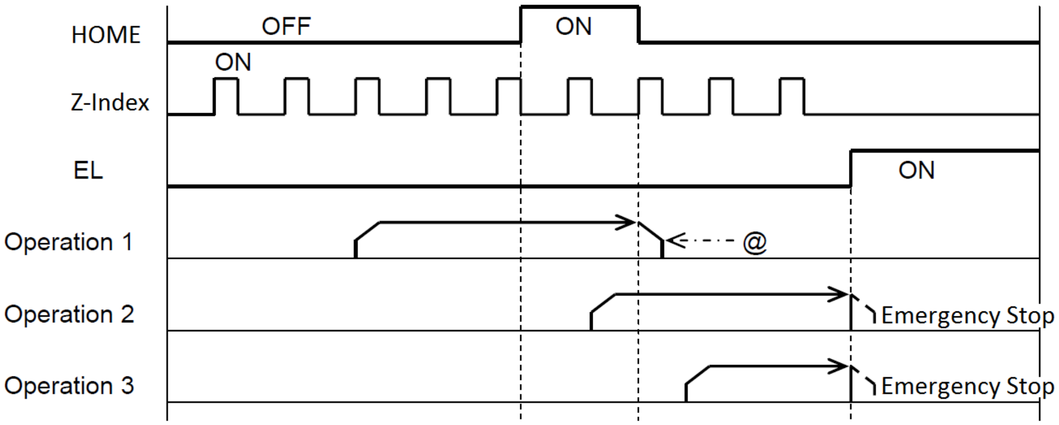
Operation 3:

A) Homing is started moving away from the home input switch.
B) Acceleration from LSPD to HSPD.
C) End limit is encountered, immediate stop.

A.4 Homing Mode 3

Homing mode 3 uses both the home limit switch and Z-index pulse. The deceleration to the home set point occurs after the move passes through the home limit switch and encounters the Z-index pulse.

Homing Mode	Home Limit	End Limits	Z-index	Slow Down	Approaches Home at		Direction to Trigger Zero	Ending Point
					High Speed	Low Speed		
Mode 3	●		●	○	H		Dir of Home	Non-Zero



Operation 1:

A) Homing operation is started, accelerate from LSPD to HSPD.
B) Continue moving at HSPD after home limit is encountered.
C) Decelerate when specified number of Z-Index pulses are counted (ZCNT), zero position is set.
D) Operation decelerates and stops.
E) ERC pulse is sent.

Operation 2:

A) Homing is started while home input is activated.
B) Acceleration from LSPD to HSPD.
C) End limit is encountered, immediate stop.

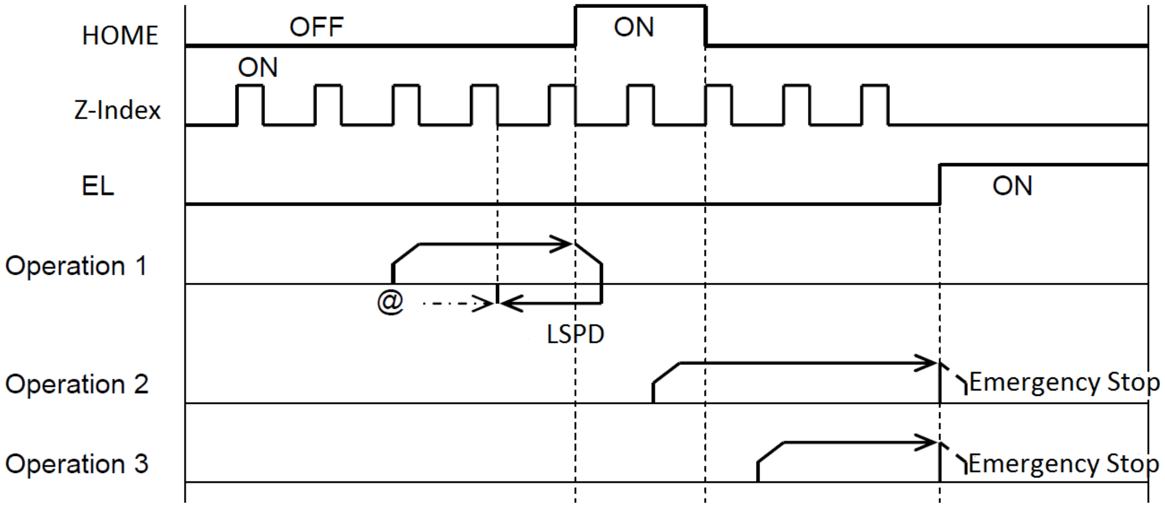
Operation 3:

A) Homing is started moving away from the home input switch.
B) Acceleration from LSPD to HSPD.
C) End limit is encountered, immediate stop.

A.5 Homing Mode 4

For homing mode 4, the motor decelerates upon reaching the home limit switch and reverses direction upon reaching the lower speed point. Home is set when the Z-index pulse is encountered.

Homing Mode	Home Limit	End Limits	Z-index	Slow Down	Approaches Home at		Direction to Trigger Zero	Ending Point
					High Speed	Low Speed		
Mode 4	●		●	○		L	Reverse Dir	Zero



Operation 1:

A) Homing operation is started, accelerate from LSPD to HSPD.
B) Begin deceleration after home limit is encountered.
C) Reverse direction at LSPD after deceleration complete.
D) Continue moving at LSPD until specified number of Z-Index pulses are counted (ZCNT).
E) Operation stops and is completed; zero position is set.
F) ERC pulse is sent.

Operation 2:

D) Homing is started while home input is activated.
E) Acceleration from LSPD to HSPD.
F) End limit is encountered, immediate stop.

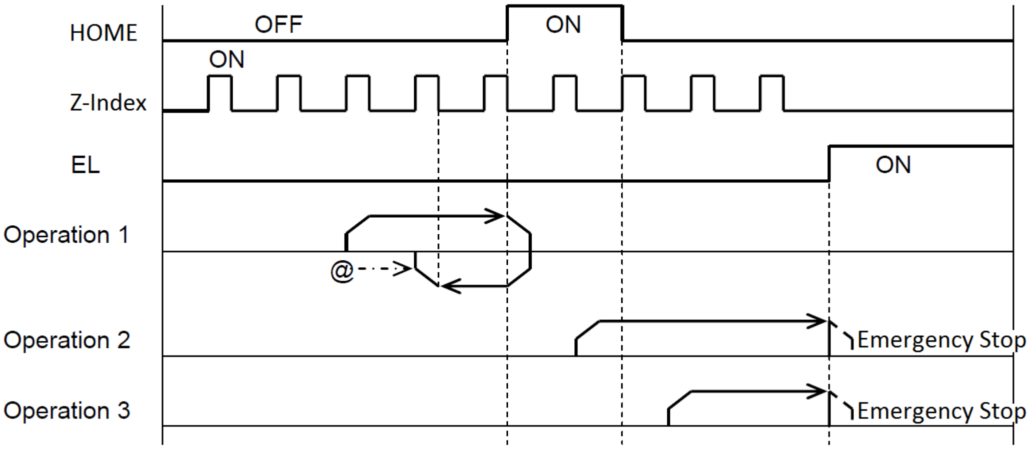
Operation 3:

D) Homing is started moving away from the home input switch.
E) Acceleration from LSPD to HSPD.
F) End limit is encountered, immediate stop.

A.6 Homing Mode 5

In homing mode 5, the home limit switch initiates a full reversal from top speed in the initial direction to top speed in the reverse direction. Homing is completed when the Z-index pulse is encountered and the motion comes to a stop.

Homing Mode	Home Limit	End Limits	Z-index	Slow Down	Approaches Home at		Direction to Trigger Zero	Ending Point
					High Speed	Low Speed		
Mode 6		●		○	H		Reverse Dir	Zero



Operation 1:

A) Homing operation is started, accelerate from LSPD to HSPD.
B) Begin deceleration to LSPD after home limit is encountered.
C) Reverse direction and accelerate to HSPD.
D) Continue moving at HSPD until specified number of Z-Index pulses are counted (ZCNT), zero position is set.
E) Operation decelerates, stops and is completed.
F) ERC pulse is sent.

Operation 2:

A) Homing is started while home input is activated.
B) Acceleration from LSPD to HSPD.
C) End limit is encountered, immediate stop.

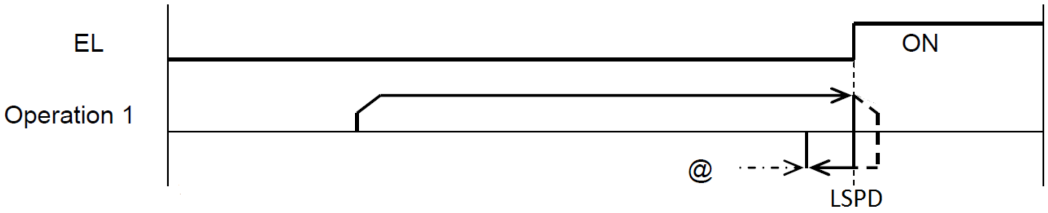
Operation 3:

A) Homing is started moving away from the home input switch.
B) Acceleration from LSPD to HSPD.
C) End limit is encountered, immediate stop.

A.7 Homing Mode 6

Homing mode 6 uses the end of travel limit switch to trigger home location. The end limit is encountered and the motor reverses off the end of travel limit at a lower speed and sets home position.

Homing Mode	Home Limit	End Limits	Z-index	Slow Down	Approaches Home at		Direction to Trigger Zero	Ending Point
					High Speed	Low Speed		
Mode 6		●		○	H		Reverse Dir	Zero



Operation 1:

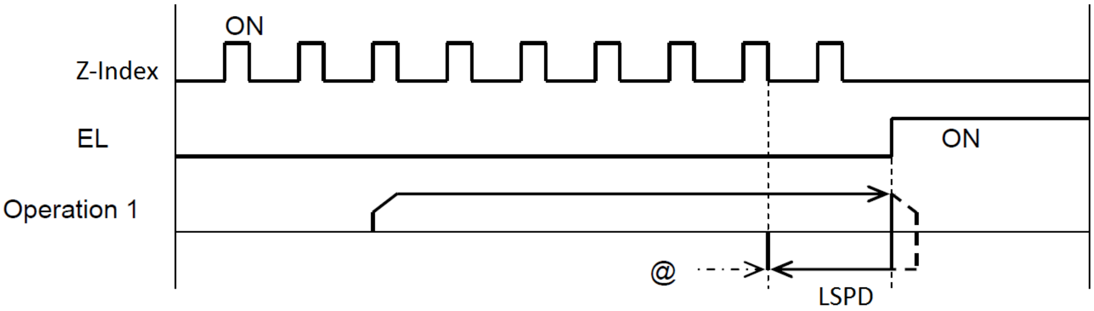
A) Homing operation is started, accelerate from LSPD to HSPD.
B) Immediate stop when the end of travel limit switch is encountered.
C) Reverse direction at LSPD until end limit switch turns off.
D) Operation stopped immediately and homing operation is completed, zero position is set.
E) ERC pulse is sent.

Note: When utilizing the end travel limits, the ERC command will clear the end-of-travel limit flag in the direction of the home during the homing operation. Refer to **ERC** command in the Command Reference for further details.

A.8 Homing Mode 7

Homing mode 7 is similar to homing mode 6, except it reverses until it locates the Z-index pulse after encountering the end of travel limit switch. Once the Z-index is reached, homing is completed.

Homing Mode	Home Limit	End Limits	Z-index	Slow Down	Approaches Home at		Direction to Trigger Zero	Ending Point
					High Speed	Low Speed		
Mode 7		●	●	○		L	Reverse Dir	Zero



Operation 1:

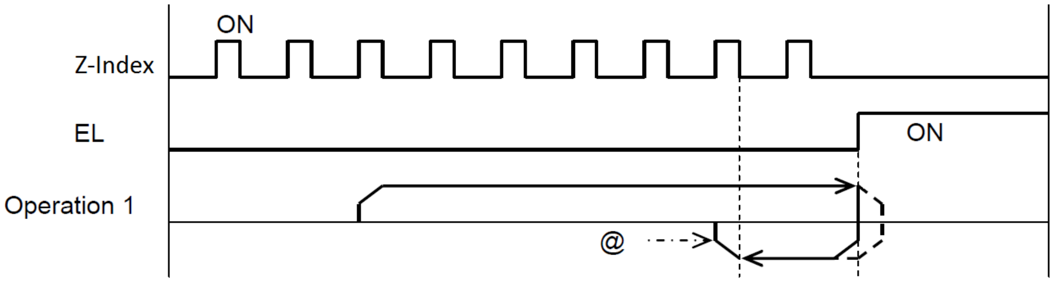
A) Homing operation is started, accelerate from LSPD to HSPD.
B) Immediate stop when the end of travel limit switch is encountered.
C) Reverse direction at LSPD until specified number of Z-Index pulses are counted (ZCNT).
D) Operation stopped immediately and homing operation is completed, zero position is set.
E) ERC pulse is sent.

Note: When utilizing the end travel limits, the ERC command will clear the end-of-travel limit flag in the direction of the home during the homing operation. Refer to **ERC** command in the Command Reference for further details.

A.9 Homing Mode 8

In homing mode 8, the end of travel limit switch triggers an immediate stop and then full reversal at top speed until Z-index is encountered and zero position is set. At this point the motor decelerates and stops.

Homing Mode	Home Limit	End Limits	Z-index	Slow Down	Approaches Home at		Direction to Trigger Zero	Ending Point
					High Speed	Low Speed		
Mode 8		●	●	○	H		Reverse Dir	Non-Zero



Operation 1:

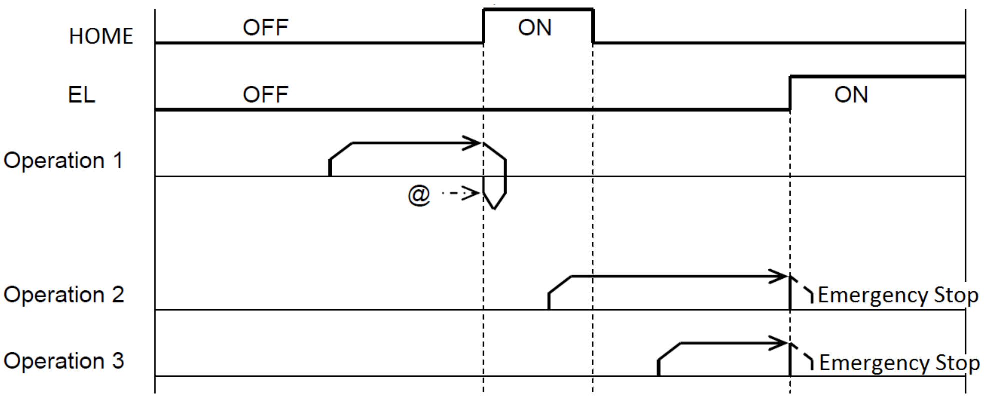
A) Homing operation is started, accelerate from LSPD to HSPD.
B) Immediate stop when the end of travel limit switch is encountered.
C) Reverse direction accelerating from LSPD to HSPD and move until specified number of Z-Index pulses are counted (ZCNT).
D) As the specified number of Z-index pulses is encountered, zero position is set and deceleration to LSPD begins.
E) Homing operation is completed by deceleration and stop.
F) ERC pulse is sent.

Note: When utilizing the end travel limits, the ERC command will clear the end-of-travel limit flag in the direction of the home during the homing operation. Refer to **ERC** command in the Command Reference for further details.

A.10 Homing Mode 9

Homing Mode 9 uses the home limit switch to initiate a deceleration, then continues with a quick index back to the position where the home limit was encountered.

Homing Mode	Home Limit	End Limits	Z-index	Slow Down	Approaches Home at		Direction to Trigger Zero	Ending Point
					High Speed	Low Speed		
Mode 9	●			○	**H**		Reverse Dir	Zero



Operation 1:

A) Homing operation is started, accelerate from LSPD to HSPD.
B) Deceleration begins as the home limit switch is encountered and zero position is set.
C) Begin zero-point return operation back to home position at the end of deceleration, accelerate from LSPD to HSPD.
D) Homing operation is completed when the 0 position is reached.
E) ERC pulse is sent.

Operation 2:

A) Homing is started while home input is activated.
B) Acceleration from LSPD to HSPD.
C) End limit is encountered, immediate stop.

Operation 3:

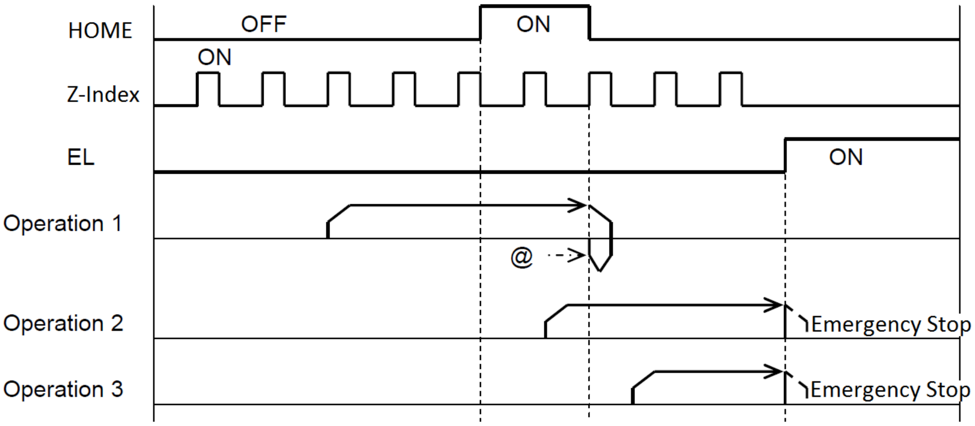
A) Homing is started moving away from the home input switch.
B) Acceleration from LSPD to HSPD.
C) End limit is encountered, immediate stop.

NOTE: For homing modes 9 – 12, encoder feedback is required. When using a stepper motor and homing with modes 9 – 12, an encoder is required and the command pulse count and encoder pulse count must be equal.

A.11 Homing Mode 10

Homing mode 10 is similar to homing mode 3 as it uses both the home limit switch and Z-index pulse. Deceleration occurs after the move passes through the home limit switch and encounters the Z-index pulse. The operation continues with a quick index back to the position where the Z-index pulse was encountered.

Homing Mode	Home Limit	End Limits	Z-index	Slow Down	Approaches Home at		Direction to Trigger Zero	Ending Point
					High Speed	Low Speed		
Mode 10	●		●	○	**H**		Reverse Dir	Zero



Operation 1:

- A) Homing operation is started, accelerate from LSPD to HSPD.
- B) Continue moving at HSPD as the home limit switch is encountered.
- C) Deceleration begins when specified number of Z-indexpulses is encountered (ZCNT), zero set.
- D) At end of deceleration, a zero-point return operation begins to the position where Z-index pulse was encountered, accelerating from LSPD to HSPD and decelerating and stopping at 0 position.
- E) Homing is completed when the 0 position is reached.
- F) ERC pulse is sent.

Operation 2:

- A) Homing is started while home input is activated.
- B) Acceleration from LSPD to HSPD.
- C) End limit is encountered, immediate stop.

Operation 3:

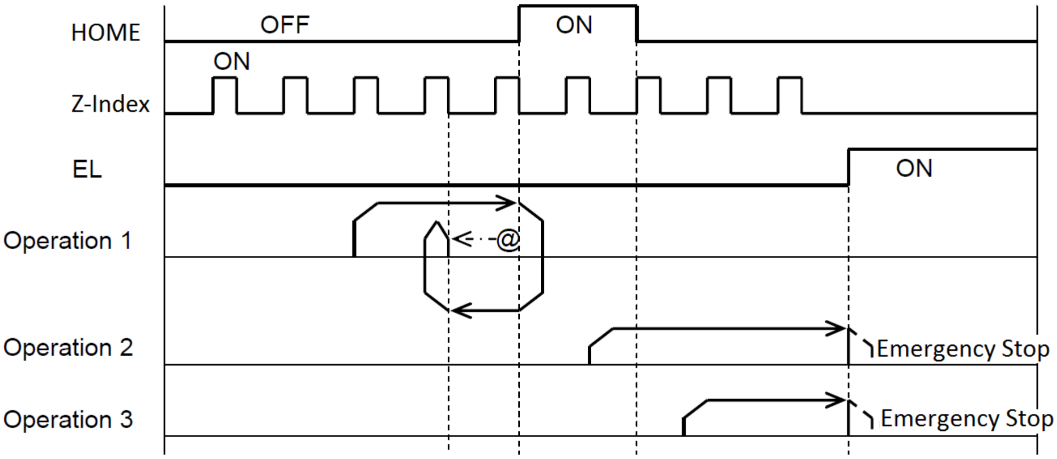
- A) Homing is started moving away from the home input switch.
- B) Acceleration from LSPD to HSPD.
- C) End limit is encountered, immediate stop.

NOTE: For homing modes 9 – 12, encoder feedback is required. When using a stepper motor and homing with modes 9 – 12, an encoder is required and the command pulse count and encoder pulse count must be equal.

A.12 Homing Mode 11

In homing mode 11, the home limit switch is used to initiate a deceleration. The operation then continues with a fast move in the reverse direction until the Z-index pulse is encountered, where a quick index reverses motion back to the position where the Z-index pulse was encountered.

Homing Mode	Home Limit	End Limits	Z-index	Slow Down	Approaches Home at		Direction to Trigger Zero	Ending Point
					High Speed	Low Speed		
Mode 11	●		●	○	H		Dir of Home	Zero



Operation 1:

A) Homing operation is started, accelerate from LSPD to HSPD, then decelerate as the home limit switch is encountered.

B) Reverse direction, accelerating from LSPD to HSPD, then decelerate when specified number of Z-indexpulses is encountered (ZCNT), zero position is set.

C) Start a zero-point return move to the position where the Z-index pulse was encountered, accelerating from LSPD to HSPD then decelerating and stopping at 0 position.

D) ERC pulse is sent.

Operation 2:

A) Homing is started while home input is activated.

B) Acceleration from LSPD to HSPD.

C) End limit is encountered, immediate stop.

Operation 3:

A) Homing is started moving away from the home input switch.

B) Acceleration from LSPD to HSPD.

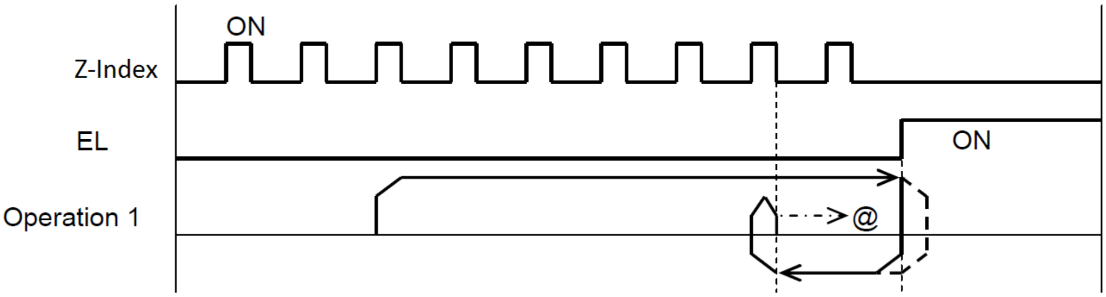
C) End limit is encountered, immediate stop.

NOTE: For homing modes 9 – 12, encoder feedback is required. When using a stepper motor and homing with modes 9 – 12, an encoder is required and the command pulse count and encoder pulse count must be equal.

A.13 Homing Mode 12

Homing mode 12 is similar to homing mode 11, except it utilizes the end of travel limit to initiate an immediate stop before it makes a fast move back to the Z-index pulse, where a quick index reverses motion back to the position where the Z-index pulse was encountered.

Homing Mode	Home Limit	End Limits	Z-index	Slow Down	Approaches Home at		Direction to Trigger Zero	Ending Point
					High Speed	Low Speed		
Mode 12		●	●	○	**H**		Dir of Home	Zero



Operation 1:

A) Homing operation is started, accelerate from LSPD to HSPD.
B) Immediate stop on encountering end of travel limit switch.
C) Reverse direction, accelerating from LSPD to HSPD.
D) Deceleration begins when specified number of Z-indexpulses is encountered (ZCNT), zero position is set.
E) Start a zero-point return move to the position where the Z-index pulse was encountered, accelerating from LSPD to HSPD then decelerating and stopping at 0 position.
F) Homing is completed when the 0 position is reached.
G) ERC pulse is sent.

NOTE: For homing modes 9 – 12, encoder feedback is required. When using a stepper motor and homing with modes 9 – 12, an encoder is required and the command pulse count and encoder pulse count must be equal.

More information on the homing modes can be found in Section 9-5-1 of the PCL6045BL users manual.

Appendix B: Speed Settings

HSPD value [PPS] †	Speed Window [SSPDM]	Min. LSPD value	Min. ACC [ms]	δ	Max ACC setting [ms]
1 - 65K	0, 1	1	2	50	((HSPD – LSPD) / δ) × 1000
65K - 130K	2	2	1	100	
130K - 325K	3	5	1	200	
325K - 650 K	4	10	1	800	
650K - 1.3M	5	20	1	1500	
1.3M - 3.2M	6	50	1	3800	
3.2M – 6.55M	7	100	1	7500	

Table B-46 Speed settings

†If StepNLoop is enabled, the HSPD range values need to be transposed from PPS (pulse/sec) to EPS (encoder counts/sec) using the following formula: **EPS = PPS / Step-N-Loop Ratio**

B.1 Acceleration/Deceleration Range

The allowable acceleration/deceleration values depend on the **LS** and **HS** settings. The minimum accel/decel setting for a given high speed and low speed is shown in the table above. The maximum accel/decel setting for a given high speed and low speed can be calculated using the formula:

Max ACC = ((HS – LS) / δ) × 1000 [ms]

Note: The ACC parameter will be automatically adjusted if the value exceeds the allowable HSPD value range

Examples:

a) If **HSPD** = 20,000 pps, **LSPD** = 10,000 pps:
   a. Min acceleration allowable: **1 ms**
   b. Max acceleration allowable: ((20,000 – 10000) / 50) x 1,000 ms = **200,000 ms** (200 sec)

b) If **HSPD** = 900,000 pps, **LSPD** = 9,000 pps:
   a. Min acceleration allowable: **1 ms**
   b. Max acceleration allowable: ((900,000 – 9,000) / 1500) x 1000 ms = **594,000** ms (594 sec)

B.2 Acceleration/Deceleration Range – Positional Move

When dealing with positional moves, the controller automatically calculates the appropriate acceleration and deceleration based on the following rules.

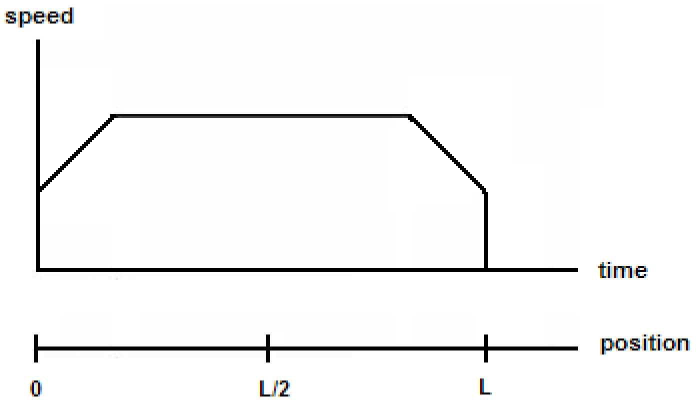


Figure B-44

1) ACC vs. DEC 1: If the theoretical position where the controller begins deceleration is less than L/2, the acceleration value is used for both ramp up and ramp down. This is regardless of the EDEC setting.
2) ACC vs. DEC 2: If the theoretical position where the controller begins constant speed is greater than L/2, the acceleration value is used for both ramp up and ramp down. This is regardless of the EDEC setting.
3) Triangle Profile: If either (1) or (2) occur, the velocity profile becomes triangle. Maximum speed is reached at L/2.

Appendix C - Interpolation

Interpolation consists of generating data points between given coordinate axis positions and using this data to generate a path in space (linear, arc, circle, or helix) related to the coordinated axes. With linear interpolation, these points establish a straight-line path through space of the associated axes. With Commander, an interpolator causes the axes to move simultaneously from the start to the end of the command. The interpolator calculates individual axis velocities to drive the axes along the desired path at the given feed rate. Thousands of intermediate coordinate points along the path are calculated between the start point and the end point of the move. Commander core utilizes linear interpolation (linear interpolation between two to four axes), arc and circular Interpolation with any two axes, and helix Interpolation with X, Y and Z axes (arc and circle (X, Y) in combination with linear interpolation (Z-axis)). Interpolation operation can be executed up to the maximum pulse frequency.

C.1 Linear Interpolation

To execute a linear interpolated move, specify the endpoint coordinates and the desired linear interpolation operation. The endpoint is specified as an incremental number of pulses from the current position on each axis, or the desired absolute position coordinates. Commander automatically identifies the axis with the larger feed distance as the main axis, and the other axes as the secondary axes. The main axis is supplied a pulse rate tied to the specified feed rate while the secondary axes are supplied with a reduced pulse rate based on the interpolation calculations. Figure C-45 shows a two-axis linear interpolation with endpoint coordinates of (10, 4) using the X and Y axes. See the PCL6045BL manual section 9.8.5, pg. 84 for more information.

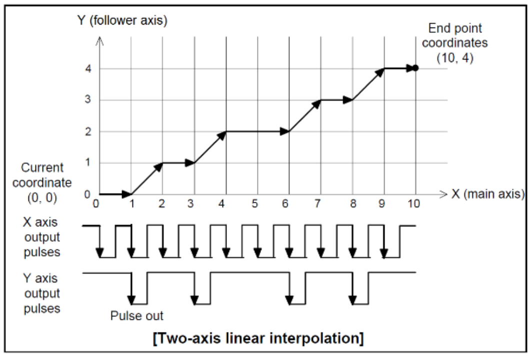


Figure C-45

C.3 Arc Interpolation

Executing an arc interpolation from the current position as the starting point requires the center coordinates and endpoint of the arc in degrees. Select either a CW or CCW arc interpolation direction and enter the arc center coordinates (incremental or absolute) and arc travel endpoint in degrees. The CW arc interpolation draws an arc from the current coordinates to the endpoint coordinates in a clockwise direction, using the center coordinates as the arc center. The CCW arc interpolation draws an arc in a counterclockwise direction. Figure C-46 is an example of drawing an arc with the X and Y axes in a CW arc interpolation operation. See the PCL6045BL manual section 9.8.8, pg. 87 for more information.

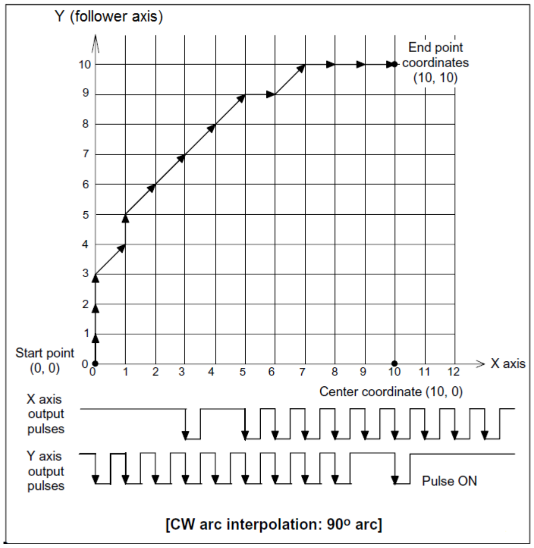


Figure C-46

C.4 Circular Interpolation

Executing circular interpolation from the current position as the starting point requires the center coordinates of the circle. Select either a CW or CCW circular interpolation direction and enter the circle center coordinates (incremental or absolute). The circular interpolation function draws a circle from the current position to the end coordinate, moving CW or CCW. The positional deviation from the specified curve is ±0.5 LSB. Figure C-47 on the right is an example of how a simple circle with a radius of 11 units is created. The LSB corresponds to the resolution of the mechanical system (size of the cells in the figure).

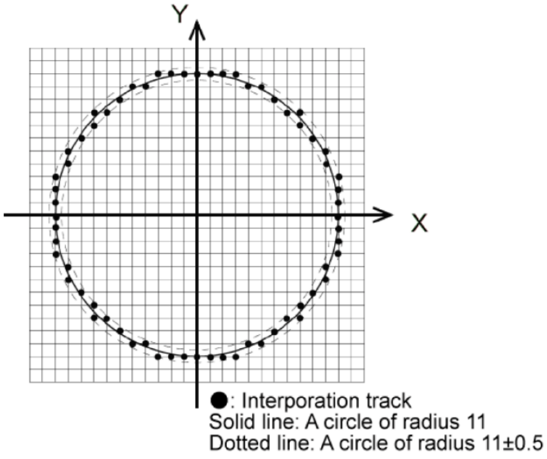


Figure C-47

C.5 Circular Interpolation Synchronized with the U-axis

The commander core allows a combination of circular or arc interpolation of X and Y axes with linear interpolation of the Z-axis to provide a helical path of motion. The U-axis is used in the background as a virtual axis to aid in the performance of this function. This feature is used in applications that may require circular interpolation between X and Y axes and to adjust the angle of a jig toward an arc tangent point with the Z-axis. Since the U-axis plays a role in these moves, it cannot be used for any other purposes.

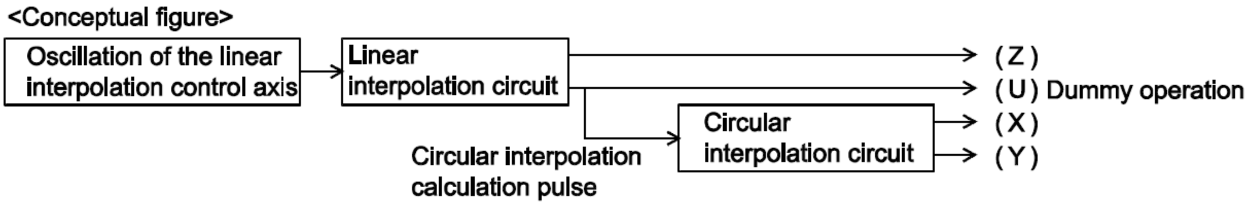

<Conceptual figure>
Oscillation of the linear interpolation control axis
Linear interpolation circuit
(Z)
(U) Dummy operation
Circular interpolation circuit
(X)
(Y)
Circular interpolation calculation pulse

Figure C-48

C.6 Additional notes regarding Interpolation functions

Acceleration/deceleration operations

Acceleration and deceleration (linear and S-curve) can be used with linear, arc, and circular interpolation operations.

Error stop

If any of the axes performing interpolated movements stops with an error, all of the axes performing interpolation will stop.

SD input

When SD input is enabled and turns ON for any axis during an interpolation operation, all axes will decelerate or decelerate and stop.

Idling control

If any axis is in idling range, none of the axes performing interpolation will accelerate.

Correction function

When a direction is changed by switching of quadrants during circular interpolation, backlash correction and slip correction control cannot be used.

Nippon Pulse America, Inc.

4 Corporate Drive Street, Radford, VA 24141 USA
Phone: 1-540-633-1677
E-mail: info@nipponpulse.com
Web: http://www. nipponpulse.com

Commander Core CMD-4CR Version 1.0
10/22/2019
P:\Temporary\CMD Manual\CMD-4CR Product manual\V1.0\Commander Manual V1.0.docx