PMX-4CX-SA

4-Axis
Stepper Motion Controller

CE RoHS 2002/95/EC

First edition, September 2005

Revision History:

1.00 – 1st Revision
1.08 – 2nd Revision
1.09 – 3rd Revision
1.10 – 4th Revision
1.11 – 5th Revision

Firmware Compatibility:

†V116BL

†If your module's firmware version number is less than the listed value, contact Arcus for the appropriate documentation.

Table of Contents

1. Introduction 5
1.1. FEATURES 5
2. Electrical Specifications 6
3. Dimensions 7
4. Connectivity 8
4.1. 2-PIN POWER CONNECTOR (5.08MM) 8
4.2. 3-PIN RS-485 CONNECTOR (3.81MM) 8
4.3. LEFT SIDE 8-PIN CONNECTOR (3.81MM) 9
4.4. LEFT SIDE 10-PIN CONNECTOR (3.81MM) 9
4.5. RIGHT SIDE 10-PIN CONNECTOR (3.81MM) 10
4.6. RIGHT SIDE 8-PIN CONNECTOR (3.81MM) 11
4.7. PMX-4CX-SA INTERFACE CIRCUIT 12
4.8. PULSE, DIRECTION, AND ENABLE OUTPUTS 13
4.9. LIMIT, HOME, AND DIGITAL INPUTS 14
4.10. DIGITAL OUTPUTS 15
5. Communication Interface 16
5.1. USB COMMUNICATION 16
5.1.1. Typical USB Setup 16
5.1.2. USB Communication API 16
5.1.3. USB Communication Issues 18
5.2. SERIAL COMMUNICATION 18
5.2.1. Typical RS-485 Setup 18
5.2.2. Communication Port Settings 19
5.2.3. ASCII Protocol 19
5.2.4. RS-485 Communication Issues 20
5.3. DEVICE NUMBER 21
5.4. WINDOWS GUI 21
6. General Operation Overview 22
6.1. MOTION PROFILE 22
6.2. PULSE SPEED 23
6.3. MOTOR POSITION 23
6.4. MOTOR POWER 23
6.5. JOG MOVE 24
6.6. STOPPING 24
6.7. POSITIONAL MOVES 25
6.8. HOMING 25
6.8.1. Home Input Only (High Speed Only) 26
6.8.2. Home Input Only (High Speed and Low Speed) 27
6.8.3. Limit Only 28
6.9. LIMITS AND ALARM SWITCH FUNCTION 28
6.10. MOTOR STATUS 29
6.11. DIGITAL INPUTS/OUTPUTS 29
6.11.1. Digital Inputs 29
6.11.2. Digital Outputs 30
6.12. POLARITY 30
6.13. COMMUNICATION TIME-OUT WATCHDOG 31
6.14. STANDALONE PROGRAM SPECIFICATION 31
6.14.1. Standalone Program Specification 31
6.14.2. Standalone Control 31
6.14.3. Standalone Status 32

6.14.4. Standalone Subroutines *32*
6.14.5. Error Handling *32*
6.14.6. Standalone Variables *33*
6.14.7. Standalone Run on Boot-Up *33*
6.15. STORING TO FLASH 34
7. Software Overview 35
7.1. MAIN CONTROL SCREEN 36
7.1.1. Status *36*
7.1.2. Control *37*
7.1.3. Digital Input/Output *38*
7.1.4. Program File Control *38*
7.1.5. Standalone Program Editor *39*
7.1.6. Standalone Program Control *39*
7.1.7. Standalone Program Compile/Download/Upload *40*
7.1.8. Setup *40*
7.1.9. Terminal *41*
7.1.10. Variable Status *42*
8. ASCII Language Specification 43
8.1. ASCII COMMAND SET 43
8.2. ERROR CODES 45
9. Standalone Language Specification 46
9.1. STANDALONE COMMAND SET 46
9.2. EXAMPLE STANDALONE PROGRAMS 48
9.2.1. Standalone Example Program 1 – Single Thread *48*
9.2.2. Standalone Example Program 2 – Single Thread *48*
9.2.3. Standalone Example Program 3 – Single Thread *49*
9.2.4. Standalone Example Program 4 – Single Thread *49*
9.2.5. Standalone Example Program 5 – Single Thread *50*
9.2.6. Standalone Example Program 6 – Single Thread *50*
9.2.7. Standalone Example Program 7 – Multi Thread *51*
9.2.8. Standalone Example Program 8 – Multi Thread *52*
A: Speed Settings 53
A.1. ACCELERATION RANGE 53
A.2. ACCELERATION RANGE – POSITIONAL MOVE 54

1. Introduction

PMX-4CX-SA is a 4 axis stepper standalone programmable motion controller.

Communication to the PMX-4CX-SA can be established over USB or RS485. It is possible to download a standalone program to the device and have it run independent of a host.

1.1. Features

- USB 2.0 communication
- RS-485 ASCII communication
  - 9600, 19200, 38400, 57600, 115200 bps
- Standalone programmable using A-SCRIPT
- Pulse/Dir/Enable open collector outputs per axis
  - Maximum pulse output rate of 400K PPS
- Opto-isolated I/O
  - 4 x inputs
  - 4 x outputs
  - +Limit/-Limit/Home inputs per axis
- Homing routines:
  - Home input only (high speed)
  - Home input only (high speed + low speed)
  - Limit only

Contacting Support

For technical support contact: support@arcus-technology.com.
Or, contact your local distributor for technical support.

2. Electrical Specifications

Parameter	Min	Max	Units
Main Power Input	+12	+24	V
	-	200	mA
Opto-supply Power Input	+12	+24	V
Pulse/Direction/Enable Open Collector	-	+24	V
	-	40	mA
Digital Input Forward Diode Current	-	90	mA
Digital Output Source Current	-	90	mA
Operating Temperature [1]	-20	+85	℃
Storage Temperature [1]	-55	+150	℃

Table 2.0

[1]Based on component ratings

3. Dimensions

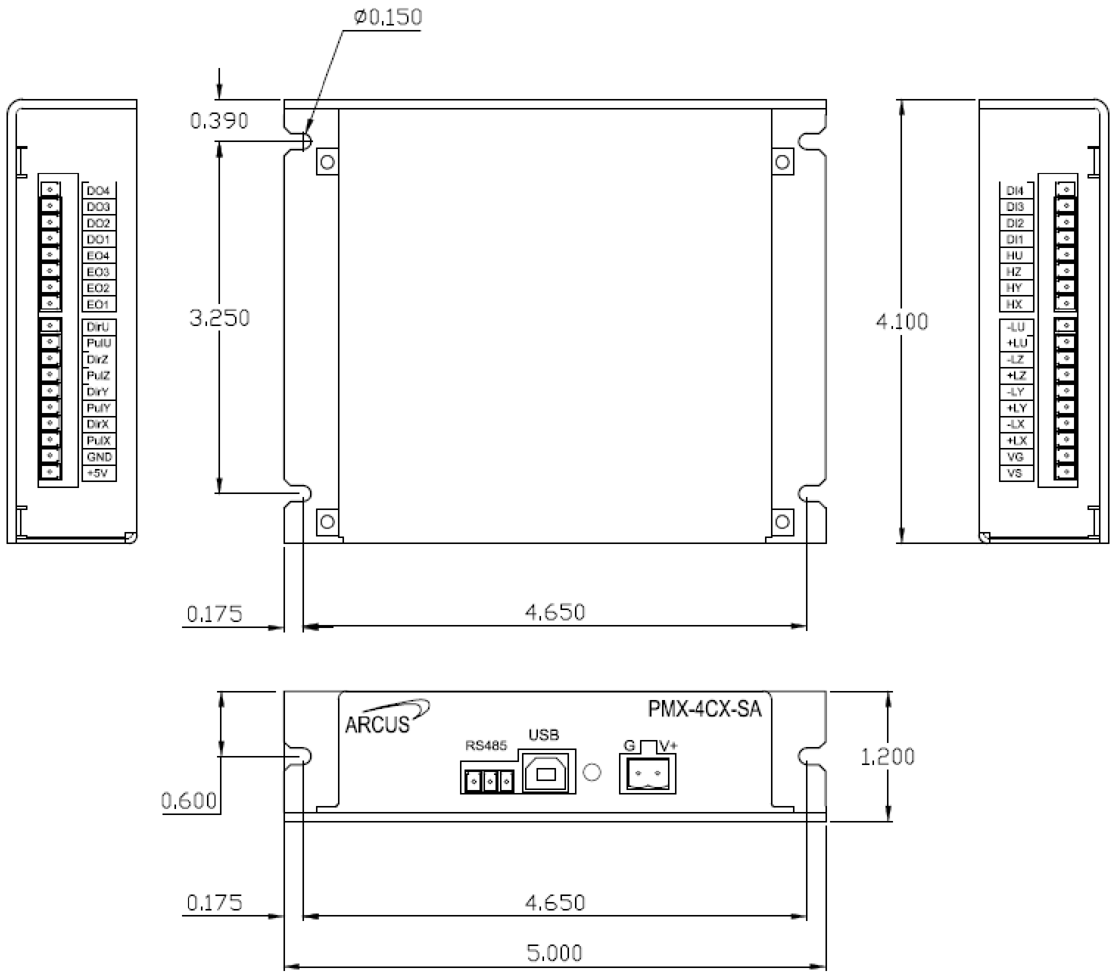


Figure 3.0

4. Connectivity

In order for PMX-4CX-SA to operate, it must be supplied with +12VDC to +24VDC. Power pins as well as communication port pin outs are shown below.

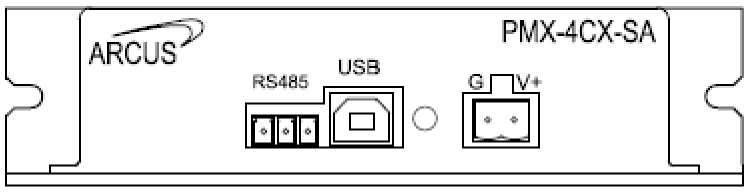


Figure 4.0

4.1. 2-Pin Power Connector (5.08mm)

Pin #	In/Out	Name	Description
1	I	G	Ground
2	I	V+	Power Input +12 to +24 VDC

Table 4.0

Mating Connector Description:	2 pin 0.2” (5.08mm) connector
Mating Connector Manufacturer:	On-Shore
Mating Connector Manufacturer Part:	[1]EDZ950/2

[1] Other 5.08mm compatible connectors can be used.

4.2. 3-Pin RS-485 Connector (3.81mm)

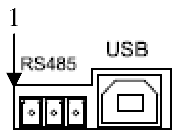


Figure 4.1

Pin #	In/Out	Name	Description
1	I	G	Ground
2	I/O	485-	RS-485 minus signal
3	I/O	485+	RS-485 plus signal

Table 4.1

Mating Connector Description:	3 pin 0.15” (3.81mm) connector
Mating Connector Manufacturer:	On-Shore
Mating Connector Manufacturer Part:	[1]EDZ1550/3

[1] Other 3.81 compatible connectors can be used.

4.3. Left Side 8-Pin Connector (3.81mm)

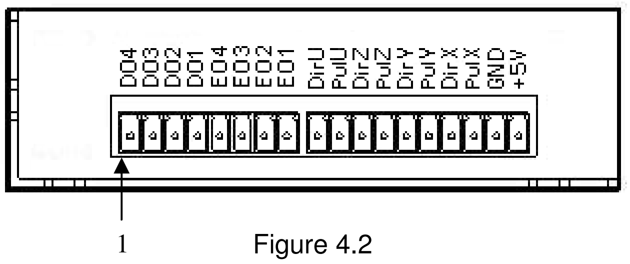


Figure 4.2

Pin #	In/Out	Name	Description
1	O	DO4	Digital Output 4
2	O	DO3	Digital Output 3
3	O	DO2	Digital Output 2
4	O	DO1	Digital Output 1
5	O	EO4	Enable Output 4 [U Axis]
6	O	EO3	Enable Output 3 [Z Axis]
7	O	EO2	Enable Output 2 [Y Axis]
8	O	EO1	Enable Output 1 [X Axis]

Table 4.2

Mating Connector Description:	Female 10 pin 2mm dual row
Mating Connector Manufacturer:	HIROSE
Mating Connector Manufacturer Part:	DF11-10DS-2C (10 pin female connector)
	DF11-2428SC (female socket pin)

4.4. Left Side 10-Pin Connector (3.81mm)

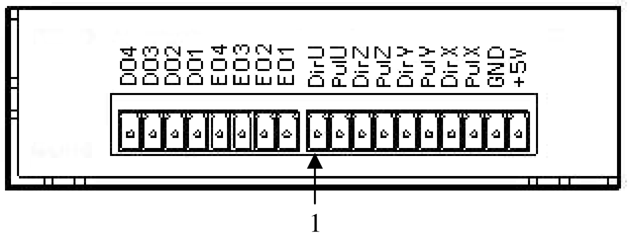


Figure 4.3

Pin #	In/Out	Name	Description
1	O	DirU	Direction [U Axis]
2	O	PulU	Pulse [U Axis]
3	O	DirZ	Direction [Z Axis]
4	O	PulZ	Pulse [Z Axis]
5	O	DirY	Direction [Y Axis]
6	O	PulY	Pulse [Y Axis]

7	O	DirX	Direction [X Axis]
8	O	PulX	Pulse [X Axis]
9	O	GND	Ground
10	O	+5V	+5 Volt Supply Output

Table 4.3

Mating Connector Description:	10 pin 0.15” (3.81mm) connector
Mating Connector Manufacturer:	On-Shore
Mating Connector Manufacturer Part:	[1]EDZ1550/10

[1]Other 3.81 compatible connectors can be used.

4.5. Right Side 10-Pin Connector (3.81mm)

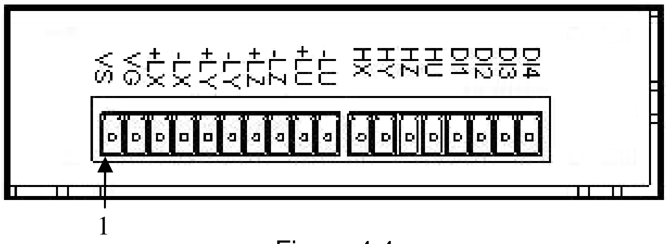


Figure 4.4

Pin #	In/Out	Name	Description
1	I	VS	Opto-supply +12-24VDC
2	I	VG	Opto-Ground
3	I	+LX	+Limit [X Axis]
4	I	-LX	-Limit [X Axis]
5	I	+LY	+Limit [Y Axis]
6	I	-LY	-Limit [Y Axis]
7	I	+LZ	+Limit [Z Axis]
8	I	-LZ	-Limit [Z Axis]
9	I	+LU	+Limit [U Axis]
10	I	-LU	-Limit [U Axis]

Table 4.4

Mating Connector Description:	14 pin 0.15” (3.81mm) connector
Mating Connector Manufacturer:	On-Shore
Mating Connector Manufacturer Part:	[1]EDZ1550/14

[1]Other 3.81 compatible connectors can be used.

4.6. Right Side 8-Pin Connector (3.81mm)

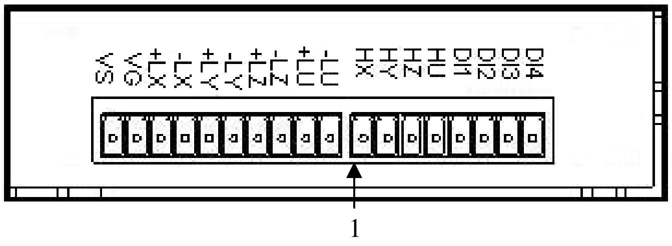


Figure 4.5

Pin #	In/Out	Name	Description
1	I	HX	Home [X Axis]
2	I	HY	Home [Y Axis]
3	I	HZ	Home [Z Axis]
4	I	HU	Home [U Axis]
5	I	DI1	Digital Input 1
6	I	DI2	Digital Input 2
7	I	DI3	Digital Input 3
8	I	DI4	Digital Input 4

Table 4.5

Mating Connector Description:	8 pin 0.15” (3.81mm) connector
Mating Connector Manufacturer:	On-Shore
Mating Connector Manufacturer Part:	[1]EDZ1550/8

[1]Other 3.81 compatible connectors can be used.

4.7. PMX-4CX-SA Interface Circuit

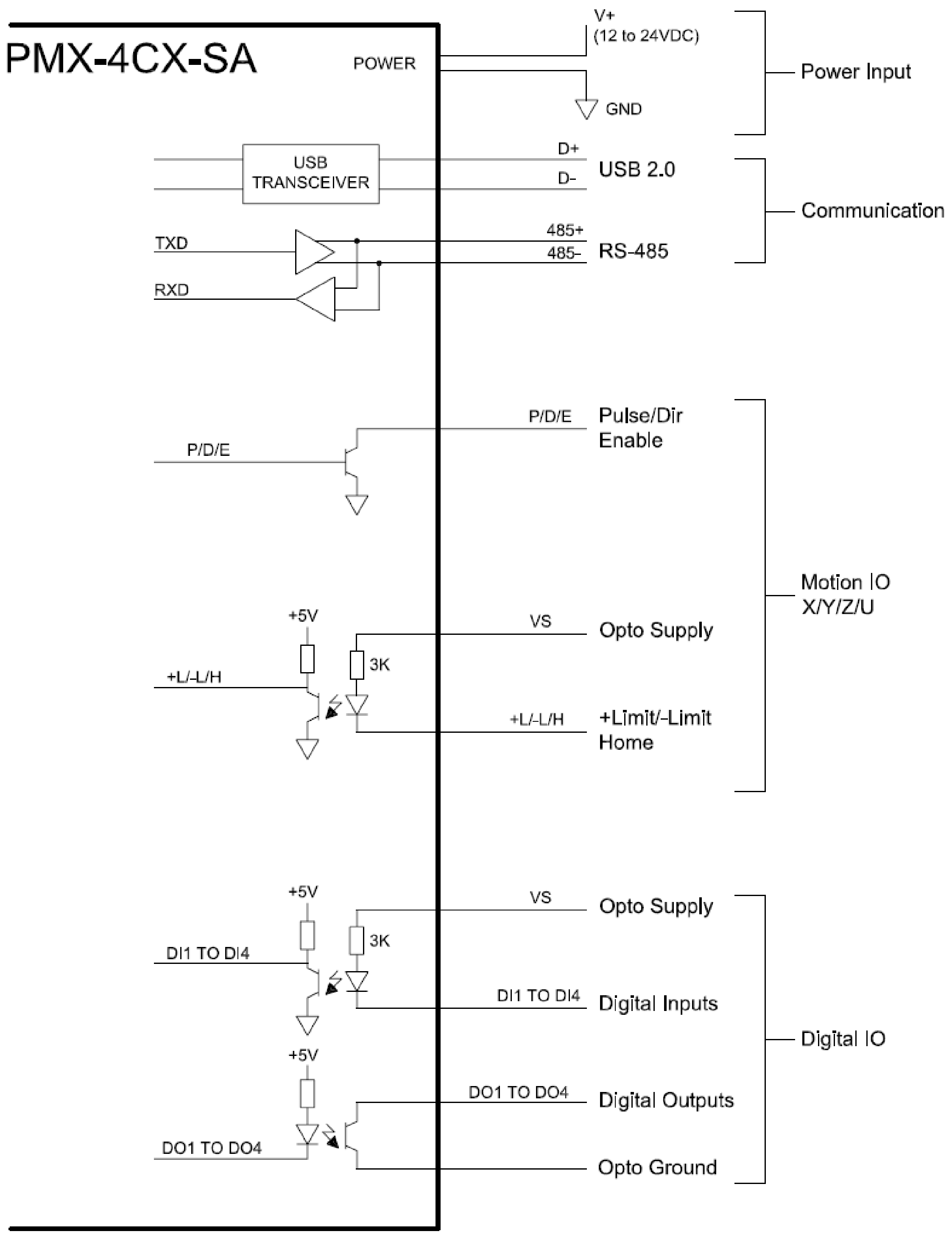


Figure 4.6

4.8. Pulse, Direction, and Enable Outputs

The Pulse, Direction, and Enable are all open collector outputs. Figure 4.7 shows the detailed schematic of these outputs. Each output is capable of sinking up to 40mA of current.

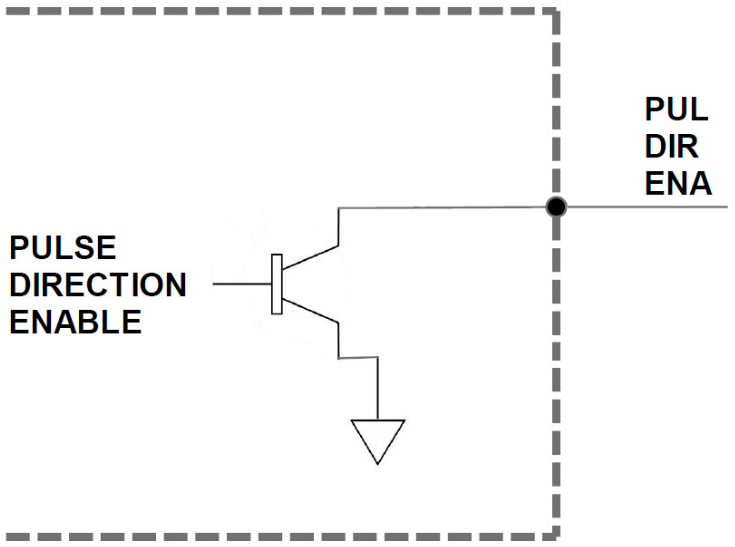


Figure 4.7

Figure 4.8 shows an example wiring diagram between the pulse, direction, and enable outputs on the PMX-4CX-SA and the corresponding input on a typical stepper driver.

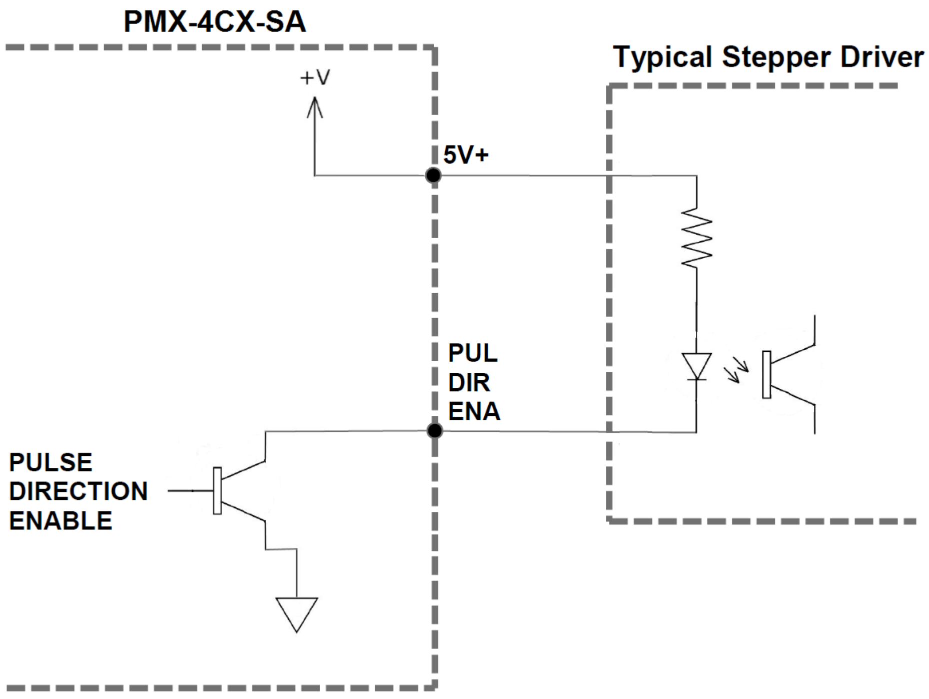


Figure 4.8

4.9. Limit, Home, and Digital Inputs

Figure 4.9 shows the detailed schematic of the opto-isolated limit, home, and general purpose digital inputs. All opto-isolated digital inputs are NPN type.

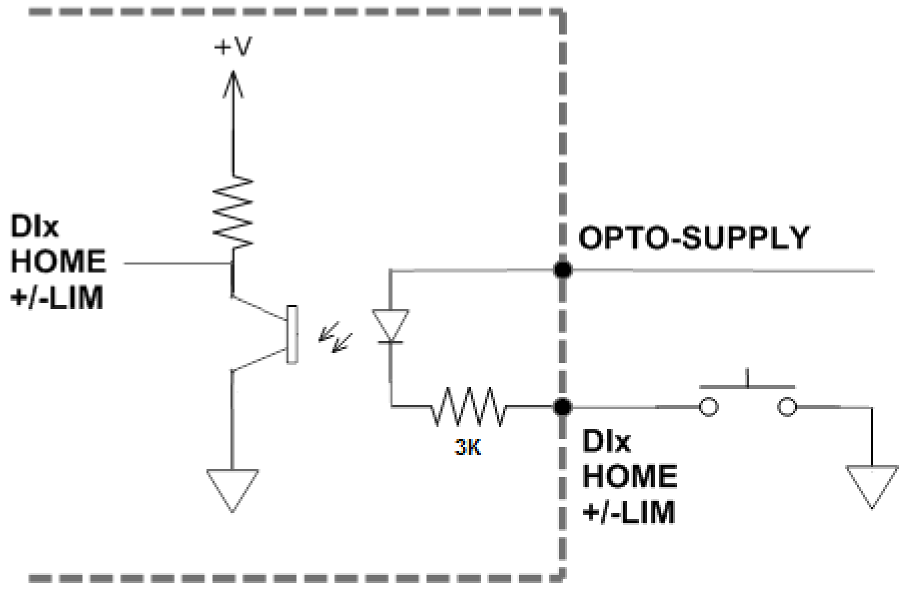


Figure 4.9

The opto-supply must be connected to +12 to +24VDC in order for the limit, home, and digital inputs to operate.

When the digital input is pulled to ground, current will flow from the opto-supply to ground, turning on the opto-isolator and activating the input.

To de-activate the input, the digital input should be left unconnected or pulled up to the opto-supply, preventing current from flowing through the opto-isolator.

4.10. Digital Outputs

Figure 4.10 shows an example wiring of the digital outputs. All opto-isolated digital outputs will be NPN type.

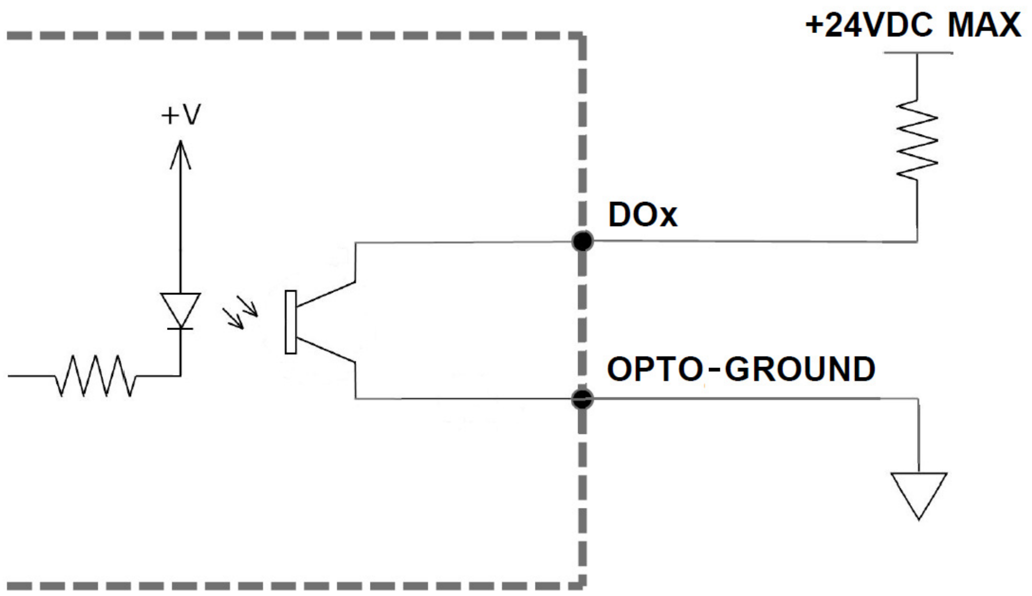


Figure 4.10

When activated, the opto-isolator for the digital output pulls the voltage on the digital output line to the power supply. The maximum sink current for digital outputs is 90mA. Take caution to select the appropriate external resistor so that the current does not exceed 90mA.

When deactivated, the opto-isolator will break the connection between the digital output and the power supply.

5. Communication Interface

5.1. USB Communication

PMX-4CX-SA USB communication is USB 2.0 compliant.

In order to communicate with PMX-4CX-SA via USB, the proper software driver must be first installed. Before connecting the PMX-4CX-SA 4-axis controller, or running any programs, please go to the Arcus web site, download the Arcus Drivers and Tools Setup, and run the installation.

All USB communication will be done using an ASCII command protocol.

5.1.1. Typical USB Setup

The PMX-4CX-SA can be connected to a PC directly via USB or through a USB hub. All USB cables should have a noise suppression choke to avoid communication loss or interruption. See a typical USB network setup in figure 5.0.

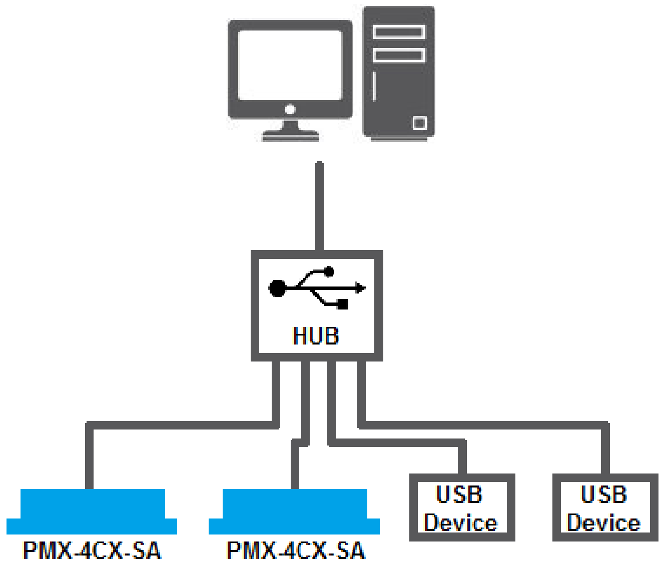


Figure 5.0

5.1.2. USB Communication API

Communication between the PC and PMX-4CX-SA is done using the Windows compatible DLL API function calls shown below. Windows programming languages such as Visual BASIC, Visual C++, LabView, or any other programming language that can use a DLL can be used to communicate with the PMX-4CX-SA.

Typical communication transaction time between PC and PMX-4CX-SA for sending a command from a PC and getting a reply from the controller using the

fnPerformaxComSendRecv() API function is in single digit milliseconds. This value will vary with CPU speed of PC and the type of command.

For USB communication, following DLL API functions are provided.

```
BOOL fnPerformaxComGetNumDevices(OUT LPDWORD lpNumDevices);
```

- This function is used to get total number of all types of Performax and Performax USB modules connected to the PC.

```
BOOL fnPerformaxComGetProductString(IN DWORD dwNumDevices,
                                    OUT LPVOID lpDeviceString,
                                    IN DWORD dwOptions);
```

- This function is used to get the Performax or Performax product string. This function is used to find out Performax USB module product string and its associated index number. Index number starts from 0.

```
BOOL fnPerformaxComOpen(IN DWORD dwDeviceNum,
                        OUT HANDLE* pHandle);
```

- This function is used to open communication with the Performax USB module and to get communication handle. dwDeviceNum starts from 0.

```
BOOL fnPerformaxComClose(IN HANDLE pHandle);
```

- This function is used to close communication with the Performax USB module.

```
BOOL fnPerformaxComSetTimeouts(IN DWORD dwReadTimeout,
                               DWORD dwWriteTimeout);
```

- This function is used to set the communication read and write timeout. Values are in milliseconds. This must be set for the communication to work. Typical value of 1000 msec is recommended.

```
BOOL fnPerformaxComSendRecv(IN HANDLE pHandle,
                            IN LPVOID wBuffer,
                            IN DWORD dwNumBytesToWrite,
                            IN DWORD dwNumBytesToRead,
                            OUT LPVOID rBuffer);
```

- This function is used to send commands and receive replies. The number of bytes to read and write must be 64 characters.

```
BOOL fnPerformaxComFlush(IN HANDLE pHandle)
```

- Flushes the communication buffer on the PC as well as the USB controller. It is recommended to perform this operation right after the communication handle is opened.

5.1.3. USB Communication Issues

A common problem that users may have with USB communication is that after sending a command from the PC to the device, the response is not received by the PC until another command is sent. In this case, the data buffers between the PC and the USB device are out of sync. Below are some suggestions to help alleviate this issue.

1) Buffer Flushing**:** If USB communication begins from an unstable state (i.e. your application has closed unexpectedly), it is recommended to first flush the USB buffers of the PC and the USB device. See the following function prototype below:

   BOOL ***fnPerformaxComFlush***(IN HANDLE pHandle)

   Note: fnPerformaxComFlush is only available in the most recent PerformaxCom.dll which is not registered by the standard USB driver installer. A sample of how to use this function along with this newest DLL is available for download on the website

2) USB Cable**:** Another source of USB communication issues may come from the USB cable. Confirm that the USB cable being used has a noise suppression choke. See figure 5.1.

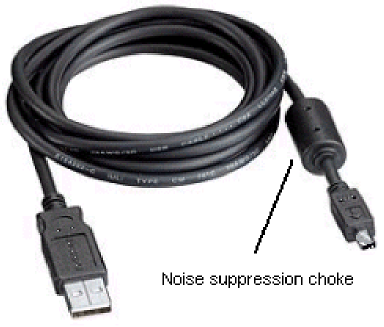


Figure 5.1

5.2. Serial Communication

The PMX-4CX-SA has the ability to communicate over an RS-485 interface using an ASCII protocol. An RS-485 serial port on the PC or PLC can be used to communicate with the PMX-4CX-SA. A USB to RS-485 converter can also be used.

5.2.1. Typical RS-485 Setup

A typical RS-485 network is shown in figure 5.2. Several techniques can be used to increase the robustness of an RS-485 network. Please see section 5.2.4 for details.

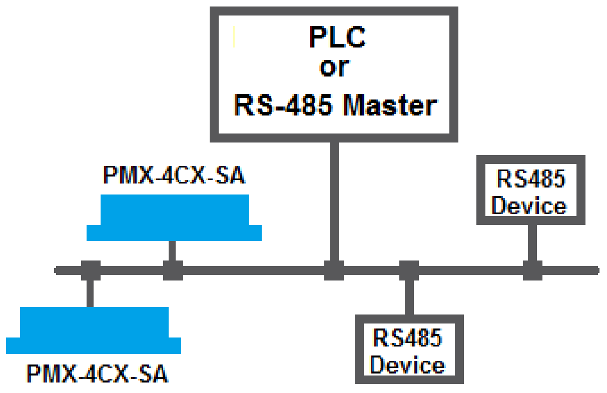


Figure 5.2

5.2.2. Communication Port Settings

The PMX-4CX-SA has the communication port settings shown in table 5.0.

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

Table 5.0

PMX-4CX-SA 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.

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

Table 5.1

To write the values to the device's flash memory, use the **STORE** command. After a complete power cycle, the new baud rate will be written to memory. Note that until a power cycle is completed, the settings will not take effect.

By default, the PMX-4CX-SA has a baud rate setting of 9600 bps.

5.2.3. ASCII Protocol

The following ASCII protocol should be used for sending commands and receiving replies from the PMX-4CX-SA. Details on valid ASCII command can be found in section 8.

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.

Examples:
    For querying the polarity
        Send: @00POL[CR]
        Reply: 7[CR]

    For reading the digital input status
        Send: @00DI[CR]
        Reply: 8[CR]

    For performing a store to flash memory
        Send: @00STORE[CR]
        Reply: OK[CR]

5.2.4. RS-485 Communication Issues

RS-485 communication issues can arise due to noise on the RS-485 bus. The following techniques can be used to help reduce noise issues.

<u>Daisy Chaining</u>
For a multi-drop RS-485 network, be sure that the network uses daisy-chain wiring. Figure 5.2 shows an example of a daisy chain network.

<u>Number of Nodes</u>
The maximum number of nodes recommended is 32. Increasing beyond this number will require special attention

<u>Twisted Pair Wiring</u>
To reduce noise, it is recommended to use twisted pair wiring for the 485+ and 485- lines. This technique will help cancel out electromagnetic interference.

<u>Termination</u>
For an RS-485 network, it may be required that a 120 Ohm resistor is placed in between the 485+ and 485- signals, at the beginning and end of the bus. A terminal resistor will help eliminate electrical reflections on the RS-485 network.

Note that on short communication buses, or buses with a small number of nodes, termination resistors may not be needed. Inclusion of terminal resistors when they are not needed may mask the main signal entirely.

5.3. Device Number

If multiple PMX-4CX-SA devices are connected to the PC, each device should have a unique device number. This will allow the PC to differentiate between multiple controllers. This is applicable for both USB and RS-485 communication types. In order to make this change to a PMX-4CX-SA, first store the desired number using the **DN** command. Note that this value must be within the range [4CX00,4CX99].

For example, to change the device number, the command **DN=4CX02** can be sent. The device name can also be changed through the *Setup* window of the standard PMX-4CX-SA software. See section 8 for details.

By default, all PMX-4CX-SA start with device number **4CX00**.

To save a modified device number to the flash memory of the PMX-4CX-SA, use the **STORE** command. After a power cycle, the new device number will be used. Note that before a power cycle is completed, the settings will not take effect.

5.4. Windows GUI

PMX-4CX-SA comes with a Windows GUI program to test, program, compile, download, and debug the controller. The Windows GUI will perform all communication via USB. See section 9 for further details.

6. General Operation Overview

Important Note: All the commands described in this section are defined as ASCII or standalone commands. ASCII commands are used when communicating over USB. Standalone commands are used when writing a standalone program onto the PMX-4CX-SA.

6.1. Motion Profile

By default, the PMX-4CX-SA uses trapezoidal velocity profile as shown in figure 6.0.

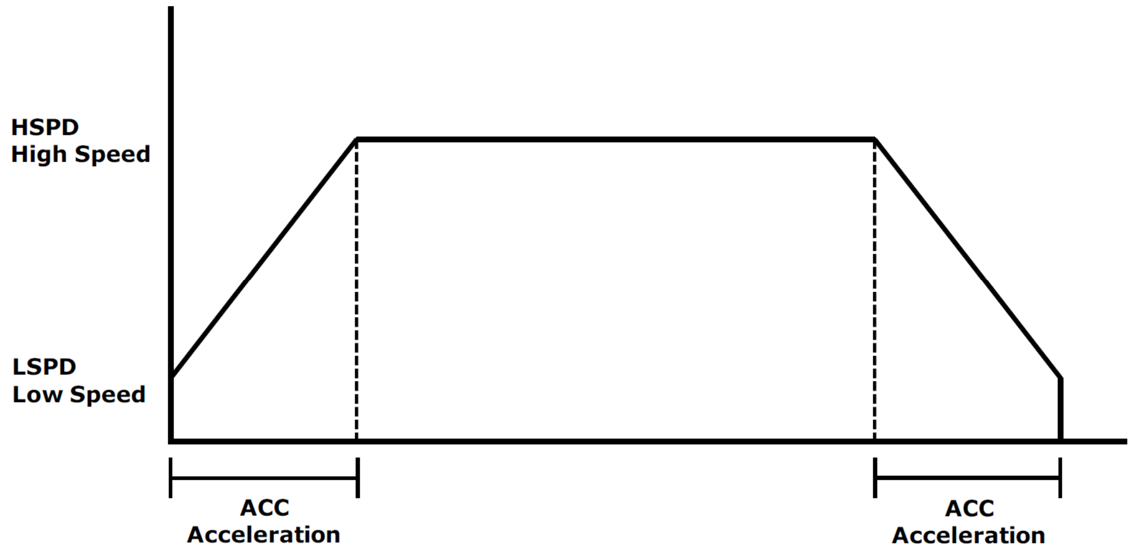


Figure 6.0

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

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

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

The minimum and maximum acceleration values depend on the high speed and low speed settings. Refer to Table A.0 and Figure A.0 in **Appendix A** for details.

By default, moves made by a single axis use global speed settings defined by **HSPD**, **LSPD**, and **ACC**. However, if a non-zero value is written to an individual speed setting, it will take priority over the global speed and will be used for single axis movements. Consider the example below.The settings below will set both the global speed settings as well as the speed setting for the X axis.

HSPD=10000
HSPDX=2000
LSPD=300
ACCX=500
ACC=300

Once a move command is issued, the global speed settings for the low speed and deceleration while using the individual X-axis speed settings for the high speed and acceleration.

ASCII	**HSPD**	**HSPX[axis]**	**LSPD**	**LSPX[axis]**	**ACC**	**ACC[axis]**
Standalone	**HSPD**	**HSPX[axis]**	**LSPD**	**LSPX[axis]**	**ACC**	**ACC[axis]**

6.2. Pulse Speed

The current pulse rate can be read in pulses/sec using the ASCII command **S[axis]**. Use the command **PS[axis]** for standalone.

ASCII	**S[axis]**
Standalone	**PS[axis]**

6.3. Motor Position

The PMX-4CX-SA has a 32 bit signed step position counter for each axis. Range of the position counter is from –2,147,483,648 to 2,147,483,647

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

ASCII	**P[axis]**
Standalone	**P[axis]**

6.4. Motor Power

The **EO** command can be used 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 enable output value must be within the 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	Description	Bit-Wise Command
0	Enable Output 1 [X-axis]	EO1
1	Enable Output 2 [Y-axis]	EO2
2	Enable Output 3 [Z-axis]	EO3
3	Enable Output 4 [U-axis]	EO4

Table 6.0

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

ASCII	**EO**	**EO[1-4]**	**EOBOOT**
Standalone	**EO**	**EO[1-4]**	**-**

6.5. Jog Move

A jog move is used to continuously move the motor without stopping. Use the **JOG[axis]+/JOG[axis]-** command when operating in ASCII or standalone mode. Once this move is started, the motor will only stop if a limit input is activated during the move or a stop command is issued.

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 8.1 for details on error responses.

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

6.6. 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 process 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]**

6.7. Positional Moves

The PMX-4CX-SA can perform positional moves for individual axis. Multiple axis can also be commanded to move to a specified position in order to perform linear coordinated motion.

The PMX-4CX-SA can perform positional moves in absolute or incremental mode. For absolute mode, the **ABS** command should be used and, for incremental mode, the **INC** command should be used. These commands should be sent before the move command is issued. The move mode will remain in absolute or incremental mode until it is changed. Use the **MM** command to read the current move mode. A return of "0" means the controller is in absolute mode while a "1" indicates incremental mode.

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

The motor status described in section 6.15 can be used to determine which move mode is active.

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.

ASCII	**ABS**	**INC**	**MM**	**X[target]**	**Y[target]**	**Z[target]**	**U[target]**
Standalone	**ABS**	**INC**	**-**	**X[target]**	**Y[target]**	**Z[target]**	**U[target]**

6.8. Homing

Home search routines involve moving the motor and using the home or limit inputs to determine the zero reference position.

The homing routines that involve a decelerated stop will result in a final position that is non-zero. The zero reference position will be preserved as the position is marked when the home trigger is detected. 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 8.1 for details on error responses.

6.8.1. Home Input Only (High Speed Only)

Use the **HOME[axis][+/-]** command. Figure 6.1 shows the homing routine.

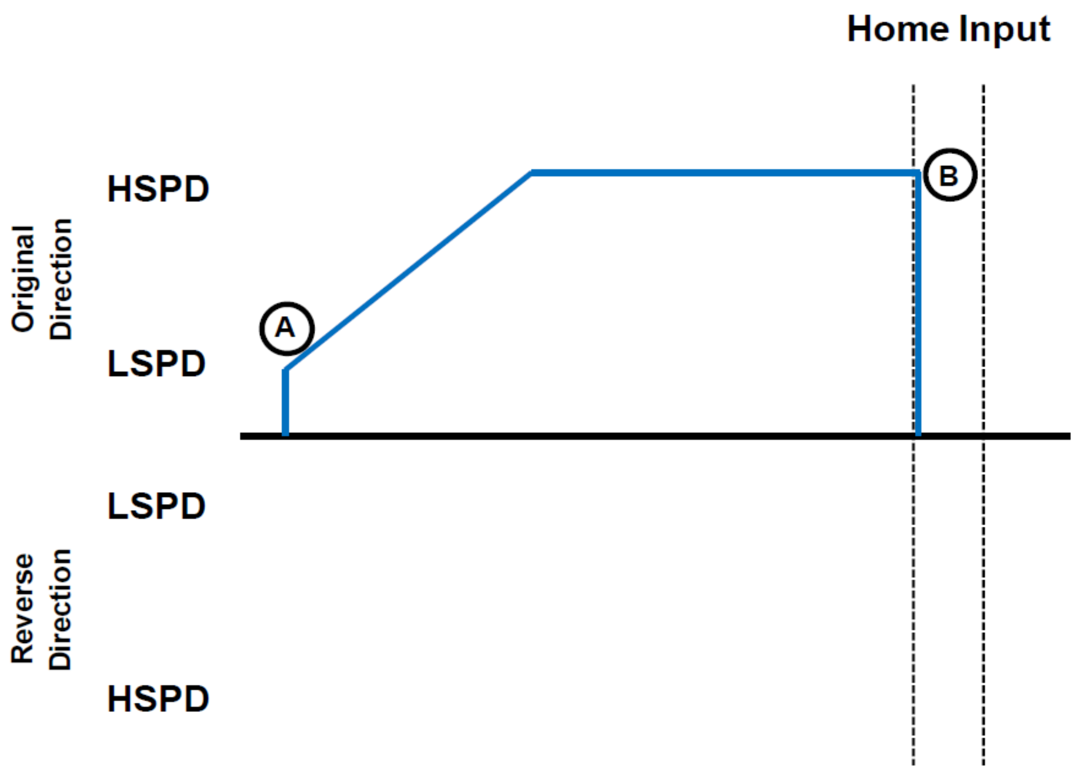


Figure 6.1

A. Issuing the command starts the motor from low speed and accelerates to high speed in search of the home input.
B. As soon as the home input is triggered, the position counter is reset to zero and the motor stops immediately.

ASCII	**HOME[axis][+/-]**
Standalone	**HOME[axis][+/-]**

6.8.2. Home Input Only (High Speed and Low Speed)

Use the **HLHOME[axis][+/-]** command. Figure 6.2 shows the homing routine.

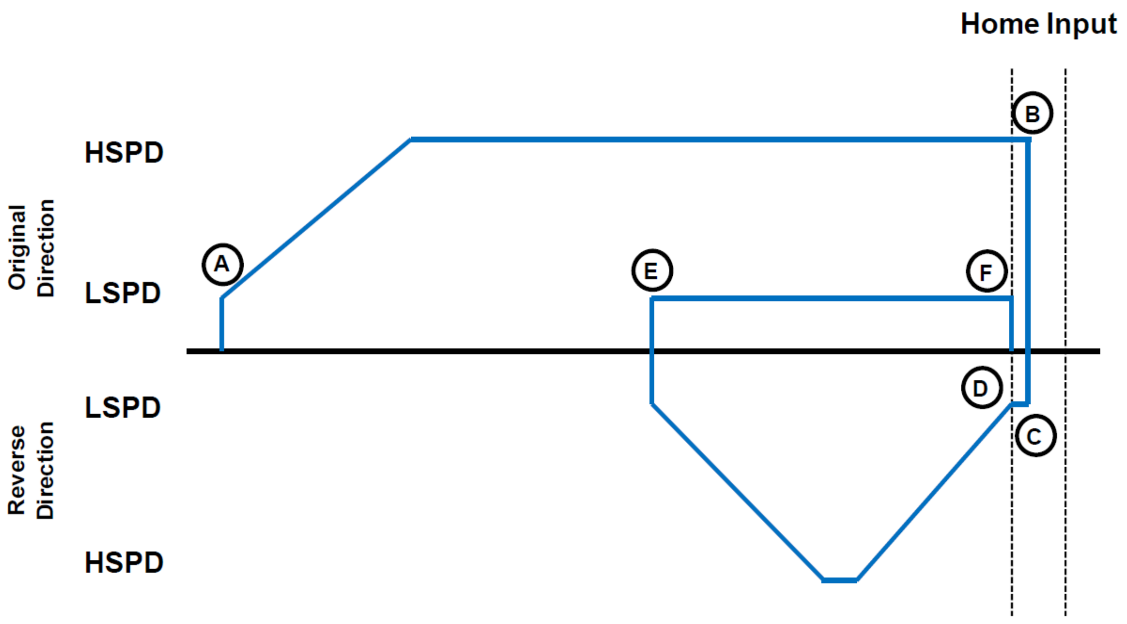


Figure 6.2

A. Starts the motor from low speed and accelerates to high speed.
B. As soon as the home input is triggered, the position counter is reset and the motor immediately stops.
C. The motor starts moving at low speed in the reverse direction until it is off the home switch.
D. It will continue past the home switch by the amount defined by the home correction amount (**HCA**). The motor will accelerate to high speed for this move.
E. The motor is now past the home input b the amount defined by the home correction amount (**HCA**). The motor now moves back toward the home switch at low speed.
F. Once the home switch is triggered again, the position counter is reset to zero and the motor immediately stops.

Note: Unique home correction amounts can also be set for each axis using the **HCA[axis]** command. If the individual value is not set, the global home correction amount will be used.

ASCII	**HLHOME[axis][+/-]**	**HCA**	**HCA[axis]**
Standalone	**HLHOME[axis][+/-]**	**-**	**-**

6.8.3. Limit Only

Use the **LHOME[axis][+/-]** command. Figure 6.3 shows the homing routine.

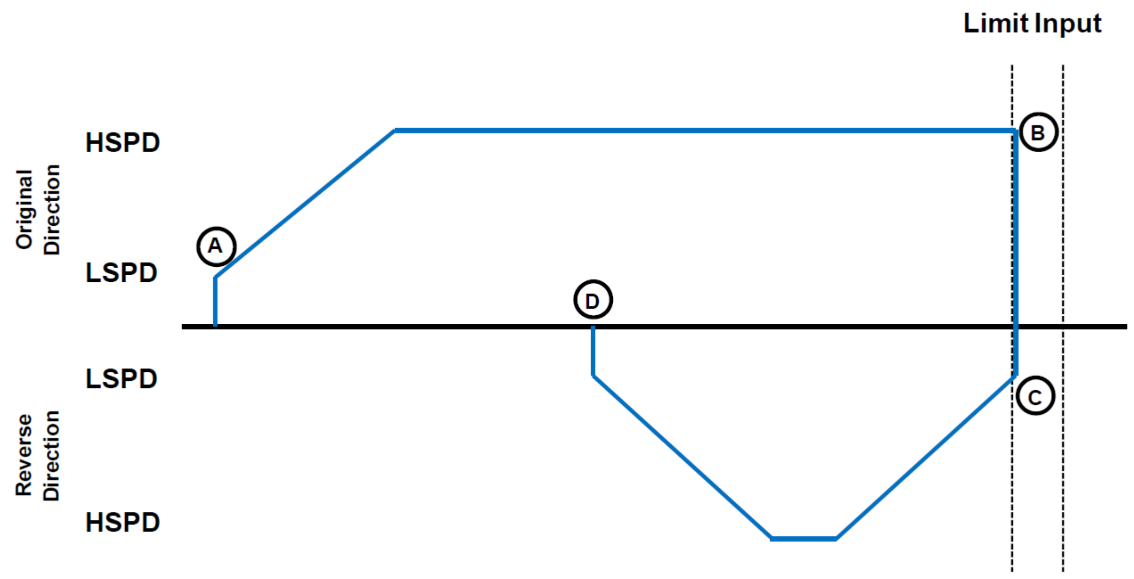


Figure 6.3

A. Issuing the command starts the motor from low speed and accelerates to high speed in search of the specified limit input.
B. As soon as the relevant limit input is triggered, the motor immediately stops motion. Depending on the high speed, it is possible that an immediate stop results in a small overshoot of the limit input.
C. The motor position is set to the limit correction amount (**LCA**) and the motor
D. The zero position is reached and the motor immediately stops.

Note: Unique limit correction amounts can also be set for each axis using the **LCA[axis]** command. If the individual value is not set, the global limit correction amount will be used.

ASCII	**LHOME[axis][+/-]**	**LCA**	**LCA[axis]**
Standalone	**LHOME[axis][+/-]**	**-**	**-**

6.9. Limits and Alarm Switch Function

Triggering the limit switch while the axis is moving will stop the motion immediately. For example, if the positive limit switch is triggered while moving in positive direction, the motor will immediately stop and the motor status bit for positive limit error is set. The same will apply for the negative limit while moving in the negative direction. Each axis will have its own designated positive and negative limit inputs.

6.10. Motor Status

Motor status can be read anytime using **MST** command. Value of the motor status is replied as an integer with following bit assignment:

Bit	Description
0	Generating pulse
1	Motor in acceleration
2	Motor in deceleration
11	TOC time-out status

Table 6.1

Motor IO status can be read anytime using the **MIO[axis]** command. Following are the bit representations of motor IO status.

Bit	Description
0	Plus limit input switch status
1	Minus limit input switch status
2	Home input switch status

Table 6.2

ASCII	**MST[axis]**	**MIO[axis]**
Standalone	**MST[axis]**	**MIO[axis]**

6.11. Digital Inputs/Outputs

PMX-4CX-SA module comes with 4 digital inputs and 4 digital outputs.

6.11.1. Digital Inputs

The digital input status of all 4 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 6.3 for details.

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

Table 6.3

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. See section 6.12 for details.

ASCII	**DI**	**DI[1-4]**
Standalone	**DI**	**DI[1-4]**

6.11.2. Digital Outputs

The **DO** command can be used to set the output voltage of all available digital outputs. The **DO** value must be within the range of 0-15.

Digital output values can also be referenced one bit 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 6.4 for details.

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

Table 6.4

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. See section 6.12 for details.

The initial state of the digital outputs can be defined by setting the **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.

ASCII	**DO**	**DO[1-4]**	**DOBOOT**
Standalone	**DO**	**DO[1-4]**	**-**

6.12. Polarity

Using the **POL** command, the polarity of the following signals can be configured.

Bit	Description
0	Digital Outputs
1	Enable Outputs
2	Digital Inputs
3	Standalone Error

Table 6.5

ASCII	**POL**
Standalone	**-**

6.13. Communication Time-out Watchdog

PMX-4CX-SA allows for the user to trigger an alarm if a master has not communicated with the device for a set period of time. When an alarm is triggered, bit 11 of the **MSTX** parameter is turned on. The time-out value is set by the **TOC** command. Units are in milliseconds. This feature is typically used in stand-alone mode. Refer to the section 9 for an example.

In order to disable this feature set **TOC=0**.

ASCII	**TOC**
Standalone	**TOC**

6.14. Standalone Program Specification

Standalone programming allows the controller to execute a user defined program that is stored in the internal memory of the PMX-4CX-SA. The standalone program can be run independently of USB and serial communication or while communication is active.

Standalone programs can be written to the PMX-4CX-SA using the Windows GUI described in section 7. Once a standalone program is written by the user, it is then compiled and downloaded to the PMX-4CX-SA. Each line of written standalone code creates 1-4 assembly lines of code after compilation

The PMX-4CX-SA can store and operate up to four separate standalone programs simultaneously.

6.14.1. Standalone Program Specification

Memory size:1,785 assembly lines ~ 10.5 KB.
Note: Each line of pre-compiled code equates to 1-4 lines of assembly lines.

6.14.2. Standalone Control

The PMX-4CX-SA supports the simultaneous execution of four standalone programs. All programs can be controlled by the **SR[0-3]** command, where Program 0 uses command **SR0**, Program 1 uses command **SR1**, and so on. For examples of multi-threading, please refer to section 9. The following assignments can be used for the **SR[0-3]** command.

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

Table 6.6

6.14.3. Standalone Status

The **SASTAT[0-3]** command can be used to determine the current status of the specified standalone program. Table 6.7 details the return values of this command.

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

Table 6.7

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 [0-1784].

6.14.4. Standalone Subroutines

The PMX-4CX-SA has the capabilities of using up to 32 separate subroutines. Subroutines are typically used to perform functions that are repeated throughout the operation of the standalone program. Note that subroutines can be shared by both standalone programs. Refer to section 9 for further details on how to define subroutines.

Once a subroutine is written into the flash, they can be called via USB communication using the **GS** command. Standalone programs can also jump to 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.

6.14.5. Error Handling

Subroutine 31 is designated for error handling. If an error occurs during standalone execution (i.e. limit error), the standalone program will automatically jump to SUB 31. If SUB 31 is not defined, the program will cease execution and go into error state. If SUB 31 is defined by the user, the code within SUB 31 will be executed.

The return jump from subroutine 31 will be determined by the bit 3 of the **POL** register. 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.

6.14.6. Standalone Variables

The PMX-4CX-SA 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[0-63]** command can be used to access the specified variables. The syntax for all available operations can be found 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 6.8

Variables V32 through V63 can be stored to flash memory using the **STORE** command. Variables V0-V31 will be initialized to zero on power up.

6.14.7. 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 table 6.9 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 6.9

Standalone programs can also be configured to run on boot-up using the Windows GUI. See section 7 for details.

ASCII	**SR[0-3]**	**SASTAT[0-3]**	**SPC[0-3]**	**GS[0-31]**	**V[0-63]**	**SLOAD**
Standalone	**SR[0-3]**	**-**	**-**	**GOSUB[0-31]**	**V[0-63]**	**-**

6.15. Storing to Flash

The following items can be stored to flash by issuing the **STORE** command.

ASCII Command	Description
DN	Device name
DB	Serial communication baud rate
DOBOOT	DO configuration at boot-up
EOBOOT	EO configuration at boot-up
HCA	Global Home Correction Amount
HCA[Axis]	Home Correction Amount for the specified axis.
LCA	Global Limit Correctiona Amount
LCA[Axis]	Limit Correction Amount for the specified axis.
POL	Polarity setting
SLOAD	Standalone program run on boot-up parameter
TOC	Time-out counter reset value
V32-V63	Note that on boot-up, V0-V31 are reset to value 0

Table 6.10

When a standalone program is downloaded, the program is immediately written to flash memory.

7. Software Overview

The PMX-4CX-SA has a Windows compatible software that allows for communication. Standalone programming, along with all other available features of the PMX-4CX-SA, will be accessible through the software. It can be downloaded from the Arcus Technology website.

To communicate over a USB connection, make sure that the PMX-4CX-SA is connected to one of the available ports on the PC. To communicate over RS485, make sure that the PMX-4CX-SA is connected to the COM port.

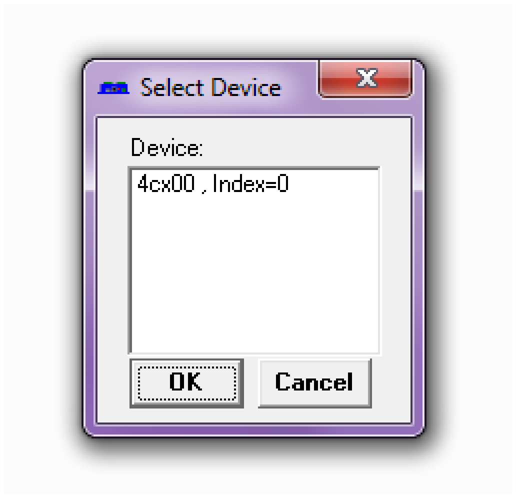
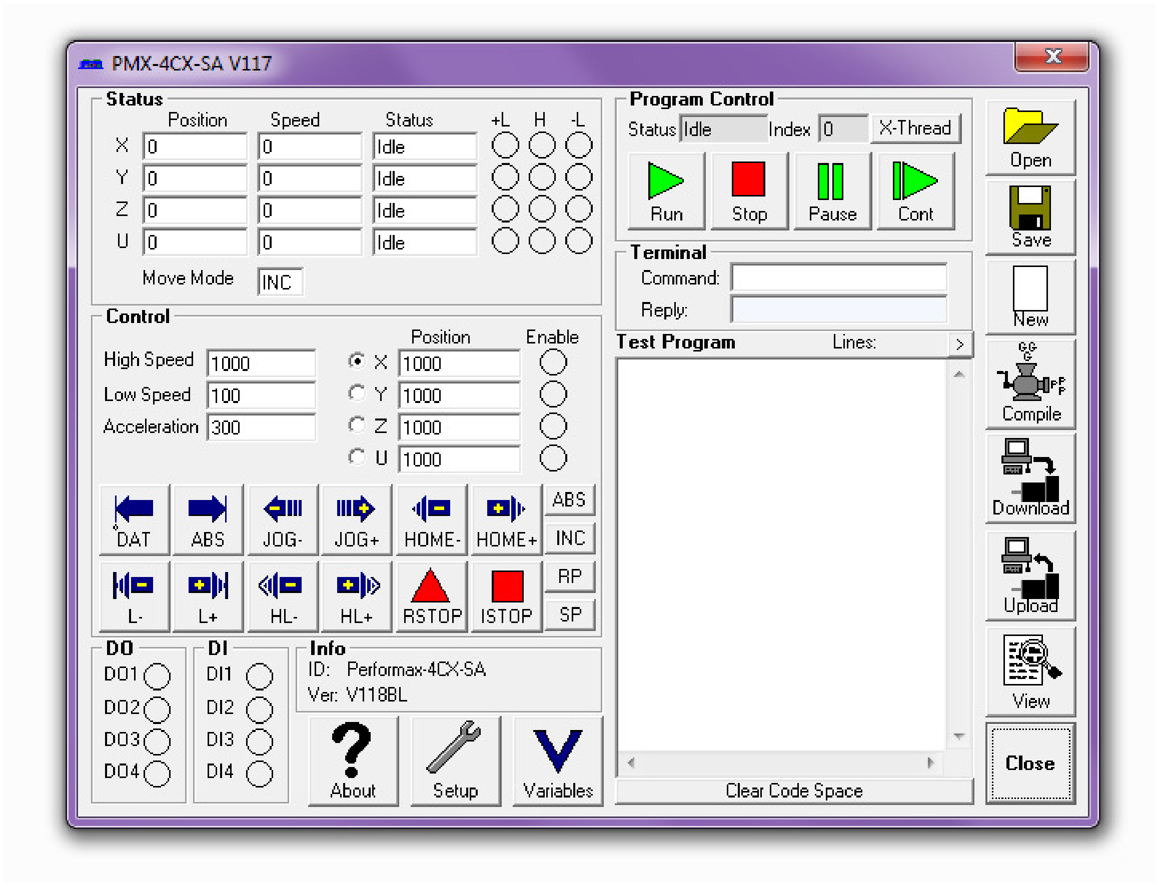


Figure 7.0

USB communication can be established by selecting the device on the start-up screen and clicking OK. All PMX-4CX-SA connected to the PC will automatically be detected and displayed.

7.1. Main Control Screen

The Main Control Screen provides accessibility to all the available functions on the PMX-4CX-SA. All features can be tested and verified.


PMX-4CX-SA V117

Status

	Position	Speed	Status	+L	H	-L
X	0	0	Idle	◯	◯	◯
Y	0	0	Idle	◯	◯	◯
Z	0	0	Idle	◯	◯	◯
U	0	0	Idle	◯	◯	◯

Move Mode: INC

Control

High Speed: 1000
Low Speed: 100
Acceleration: 300

	Position	Enable
◉ X	1000	◯
◯ Y	1000	◯
◯ Z	1000	◯
◯ U	1000	◯

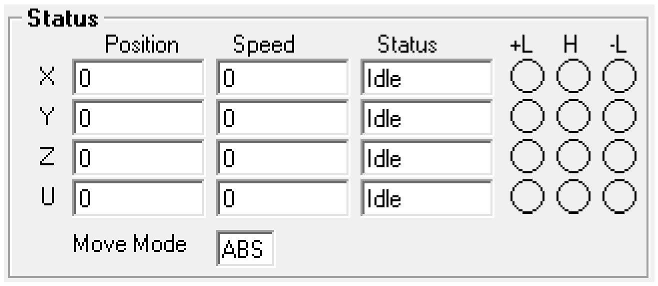
Program Control: Status: Idle; Index: 0

Terminal: Command: ; Reply:

Info: ID: Performax-4CX-SA; Ver: V118BL

Figure 7.1

7.1.1. Status


Status

	Position	Speed	Status	+L	H	-L
X	0	0	Idle	◯	◯	◯
Y	0	0	Idle	◯	◯	◯
Z	0	0	Idle	◯	◯	◯
U	0	0	Idle	◯	◯	◯

Move Mode: ABS

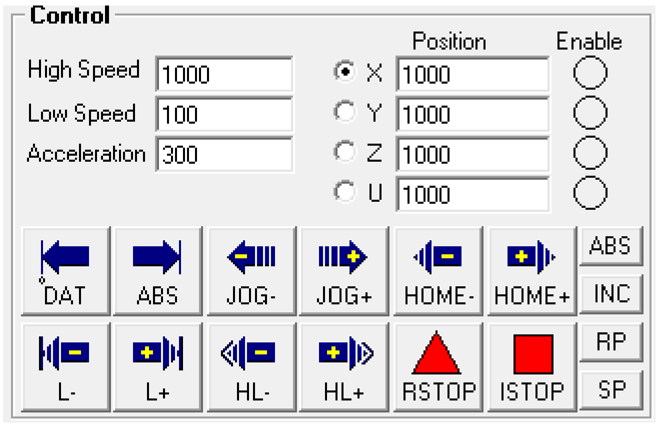
Figure 7.2

1. **Position** (X,Y,Z,U) - displays the current pulse position counter.
2. **Speed** (X,Y,Z,U) - displays the current pulse speed output rate.
3. **Motor Status** (X,Y,Z,U axes) - the current motor status of the axis. The following status can be shown.

- Idle – motor is not moving.
- Accel – motor is accelerating
- Const – motor is running in constant speed
- Decel – motor is decelerating

4. **Limit/Home/Alarm Input Status** (X,Y,Z,U axes) - Indicators for the limit and home inputs.
5. **Move Mode** - displays the current move mode for positional moves.
   - ABS – absolute move
   - INC – incremental move

7.1.2. Control


Control

			Position	Enable
High Speed	1000	(●) X	1000	◯
Low Speed	100	( ) Y	1000	◯
Acceleration	300	( ) Z	1000	◯
		( ) U	1000	◯

Figure 7.3

1. **High/Low Speed, and Accel** - use these to set the speed of a move command. To give each axis individual speed parameters, enter HSPD[axis], LSPD[axis], and ACC[axis] commands via the command terminal.
2. **X/Y/Z/U Bubbles** - move performs by one of the move buttons will apply to the selected axis.
3. **Target** (X,Y,Z,U) - positional moves will use this value as the target position.
4. **Enable** (X,Y,Z,U) – motor power is turned on or off by clicking on these circles.
5. **ABS** - Set absolute move mode
6. **INC** - Set incremental move mode
7. **RP** - Reset pulse counter for the specified axis. Not allowed if StepNLoop is enabled.
8. **SP** - Set pulse counter for the specified axis. This will use the value in the Target position field.
9. **ISTOP** – the motion is immediately stopped without deceleration.
10. **RSTOP** – the motion is stopped with deceleration.
11. **HOME+/-** - Home the axis at high speed using only the home input.

12. **HL+/-** -Home the axis at high speed and low speed using only the home sensor.
13. **L+/L-** - Home the axis using only the limit sensor.
14. **JOG+/JOG-** - Move the axis indefinitely in the positive or negative direction.
15. **ABS** - Perform absolute move. If more than one axis is selected, an interpolated move will result.
16. **DAT** - Return to 0 position. If more than one axis is selected, an interpolated move will result.

7.1.3. Digital Input/Output

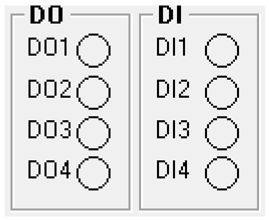


Figure 7.4

1. **DO** - displays the current status of DO1-DO4. To turn on/off a digital output, click on the corresponding circle.
2. **DI** - displays the current status of DI1-DI4.

7.1.4. Program File Control

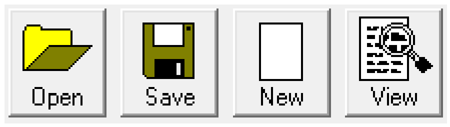


Figure 7.5

1. **Open** – Open standalone program
2. **Save** – Save standalone program
3. **New** – Clear the standalone program editor
4. **View** – View the compiled program

7.1.5. Standalone Program Editor

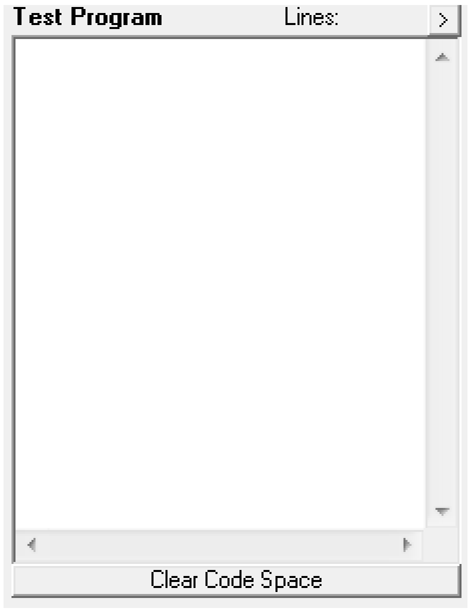


Figure 7.6

1. **Text Program** – Text box for writing and editing a standalone program.
2. **Clear Code Space** – Clear the code space on the PMX-4CX-SA.

7.1.6. Standalone Program Control

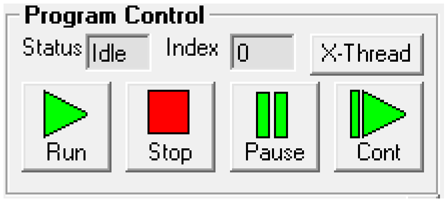


Figure 7.7

1. **Run** – run the standalone program (PRG 0)
2. **Stop** – stop the standalone program (PRG 0)
3. **Pause** – pause the standalone program (PRG 0)
4. **Cont** – continue the paused standalone program (PRG 0)
5. **XThread** – open the Standalone Program Control. Will allow for control of all 4 standalone programs.
6. **Index** – the current line of low-level code that is being executed.
7. **Status** – the current status of the standalone program (PRG 0)
    - Idle – Program is not running.
    - Running – Program is running.
    - Paused – Program is paused.
    - Error – Program is in an error state.

7.1.7. Standalone Program Compile/Download/Upload

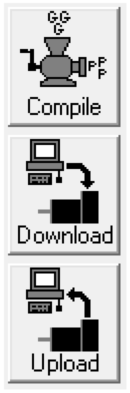
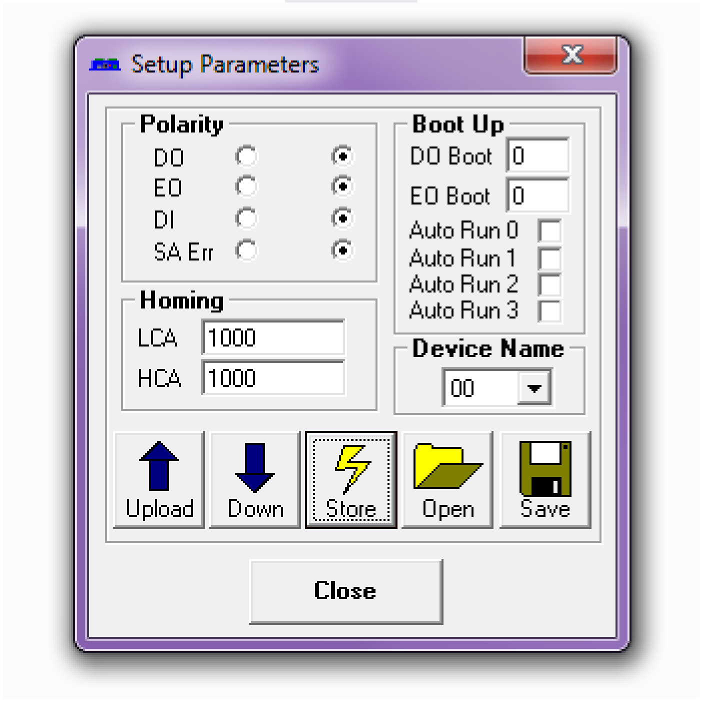


Figure 7.8

1. **Compile** – Compile the standalone program
2. **Download** – Download the compiled program
3. **Upload** – Upload the standalone program from the controller

7.1.8. Setup


Setup Parameters

Polarity
DO ○ ●
EO ○ ●
DI ○ ●
SA Err ○ ●

Homing
LCA 1000
HCA 1000

Boot Up
DO Boot 0
EO Boot 0
Auto Run 0 ☐
Auto Run 1 ☐
Auto Run 2 ☐
Auto Run 3 ☐

Device Name
00

Figure 7.9

1. **Polarity Settings**
    a. **DO** - set the digital output polarity
    b. **EO** - set the enable output polarity
    c. **DI** - set the digital input polarity
    d. **SA Err** - set the return jump line for standalone error handling
2. **Communication**
    a. **Device Name** - set device name: [4CX00-4CX99]
    b. **Baud Rate (RS485)** - set baud rate (used for RS-485 communication)
3. **Bootup Parameters**
    a. **Auto Run N** - start the selected standalone program on bootup.
    b. **DOBOOT**/**EOBOOT** - set the digital and enable output configuration status on boot-up.
4. **Open**/**Save** parameters to file.
5. **Upload**/**Download** parameters to and from flash memory.

7.1.9. Terminal

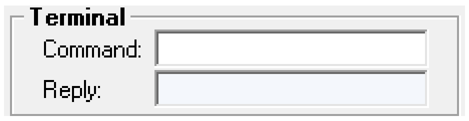
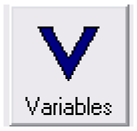
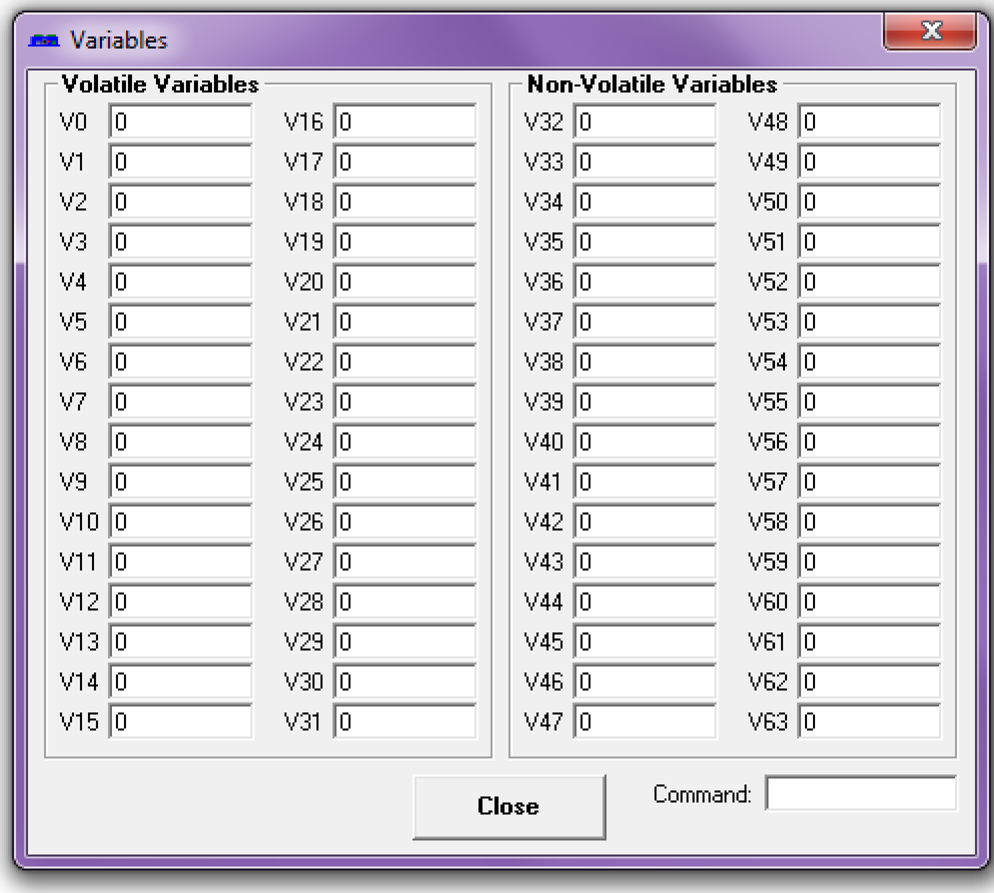


Figure 7.10

1. **Command** Send a command to the PMX-4CX-SA through this terminal.
2. **Reply** Replies from the PMX-4CX-SA will be shown here.

7.1.10. Variable Status





Volatile Variables

V0	0	V16	0
V1	0	V17	0
V2	0	V18	0
V3	0	V19	0
V4	0	V20	0
V5	0	V21	0
V6	0	V22	0
V7	0	V23	0
V8	0	V24	0
V9	0	V25	0
V10	0	V26	0
V11	0	V27	0
V12	0	V28	0
V13	0	V29	0
V14	0	V30	0
V15	0	V31	0

Non-Volatile Variables

V32	0	V48	0
V33	0	V49	0
V34	0	V50	0
V35	0	V51	0
V36	0	V52	0
V37	0	V53	0
V38	0	V54	0
V39	0	V55	0
V40	0	V56	0
V41	0	V57	0
V42	0	V58	0
V43	0	V59	0
V44	0	V60	0
V45	0	V61	0
V46	0	V62	0
V47	0	V63	0

Figure 7.11

1. **Volatile Variables** – status of volatile variable V0-V31
2. **Non-volatile Variables** – status of non-volatile variable V32-V63
3. **Command line** – set variables using V[0-63]=[value] syntax

8. 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 9 for details regarding standalone commands.

PMX-4CX-SA ASCII protocol is case sensitive. All commands should be in upper case letters. The [axis] value in the relevant commands below can be "X", "Y", "Z", or "U".

An invalid command is returned with a "?". Always check for the proper reply when a command is sent.

For USB and RS-485 communication, the commands detailed in table 8.0 are valid.

8.1. ASCII Command Set

Command	Description	Return
ABORT	Immediately stops all axis if in motion. Abort turns off the buffered move.	OK
ABORT[axis]	Immediately stops the individual axis if in motion. Abort turns off the buffered move.	OK
ABS	Set the move mode to absolute mode.	OK
ACC	Return the current global acceleration in milliseconds.	milliseconds
ACC=[value]	Set global acceleration in milliseconds.	OK
ACC[axis]	Return current individual acceleration in milliseconds.	milliseconds
ACC[axis]=[value]	Set individual acceleration in milliseconds.	OK
DB	Return the current baud rate or the controller.	Refer to Table 5.1
DB=[value]	Set baud rate of the controller.	OK
DI	Return the status of the digital inputs.	Refer to Table 6.3
DI[1-4]	Return the bit status of general purpose digital input.	[0,1]
DO	Return the status of the digital outputs.	Refer to Table 6.4
DO=[value]	Set the digital outputs. Refer to Table 6.4.	OK
DO[1-4]	Return status of individual digital output.	[0,1]
DO[1-4]=[value]	Set the individual digital output. Refer to Table 6.4.	OK
DOBOOT	Return the DO configuration at boot-up.	[0-255]
DOBOOT=[value]	Set the DO configuration at boot-up.	OK
DN	Return the device name.	[4CX00-4CX99]
DN=[value]	Set the device name.	OK
EO	Returns the enable output status.	Refer Table 6.0
EO=[value]	Set the enable outputs.	OK
EO[1-4]	Return the individual enable output status.	[0,1]
EO[1-4]=[value]	Set the individual enable output.	OK
EOBOOT	Return the EO configuration at boot-up.	Refer to section

		6.4
EOBOOT=[value]	Set the EO configuration at boot-up.	OK
GS[0-31]	Call a subroutine that has been previously stored to flash memory.	OK
HSPD	Return the global high speed setting.	[1-6,000,000]PPS
HSPD=[value]	Set the global high speed.	OK
HSPD[axis]	Return the individual high speed setting.	[1-6,000,000]PPS
HSPD[axis]=[value]	Sets individual high speed.	OK
HCA	Return the global home correction amount.	32-bit number
HCA[axis]	Return the home correction amount for the indicated axis	32-bit number
HCA=[Value]	Set the global home correction amount	OK
HCA[axis]=[Value]	Set the home correction amount for the indicated axis	OK
HLHOME[axis][+/-]	Home the indicated axis at low and high speed in the positive/negative direction	OK
HOME[axis][+/-]	Home the indicated axis in the positive/negative direction	OK
ID	Return the product ID.	Performax-4CX-SA
INC	Set the incremental move mode.	OK
JOG[axis][+/-]	Jogs the axis in plus [+] or minus [-] direction.	OK
LCA	Return the global limit correction amount.	32-bit number
LCA[axis]	Return the limit correction amount for the indicated axis	32-bit number
LCA=[Value]	Set the global limit correction amount	OK
LHOME[axis][+/-]	Home the indicated axis using the limit inputs in the positive/negative direction.	OK
LSPD	Return the global low speed setting..	[1-6,000,000]
LSPD=[value]	Set the global low speed.	OK
LSPD[axis]	Return the individual low speed setting.	[1-6,000,000]
LSPD[axis]=[value]	Set the individual low speed.	OK
MM	Returns the controller move mode	0 - ABS mode 1 - INC mode
MST[axis]	Returns motor status for the specified axis.	Refer to Table 6.1
MIO[axis]	Returns motor IO status for the specified axis.	Refer to Table 6.2
POL	Return the current polarity setting.	Refer to Table 6.5
POL=[value]	Set the polarity setting. Refer to Table 6.5.	OK
P[axis]	Return the position value of the individual axis.	28-bit number
P[axis]=[value]	Set the position value of the individual axis.	OK
S[axis]	Returns the actual pulse rate for the specified axis.	PPS
SA[0-1784]	Return the standalone line.	
SA[0-1784]=[value]	Set the standalone line.	OK
SASTAT	Return the standalone program status. Refer to Table 6.7.	[0-4]
SLOAD	Returns the standalone program run on boot	[0-15]

	parameter. Refer to Table 6.9.	
SLOAD=[value]	Set the standalone program run on boot parameter. Refer to Table 6.9.	OK
SR[0-3]=[value]	Control the standalone program. Refer to Table 6.6.	OK
SPC[0-3]	Returns the program counter for the specified standalone program.	[0-1784]
STOP	Performs ramp down to stop for all axis if in motion.	OK
STOP[axis]	Performs ramp down to stop for individual axis.	OK
STORE	Store settings to flash. Refer to Table 6.10.	OK
TOC	Returns the communication time-out parameter in milliseconds.	milliseconds
TOC=[value]	Set the communication time-out parameter in milliseconds.	OK
V[0-63]	Return the standalone variable value.	32-bit number
V[0-63]=[value]	Set the standalone variable value.	OK
VER	Return the controller firmware version	V[#]
X[target X] Y[target Y] Z[target Z] U[target U]	Perform an individual/interpolated move	OK

Table 8.0

8.2. Error Codes

If an ASCII command cannot be processed by the PMX-4CX-SA, the controller will reply with an error code. See below for possible error responses:

Error Code	Description
?[Command]	The ASCII command is not understood by the PMX-4CX-SA
?Index out of Range	The index for the command sent to the controller is not valid.
?Moving	A move or position change command is sent while the PMX-4CX-SA is outputting pulses.
?Sub not Initialized	Call to a subroutine using the **GS** command is not valid because the specified subroutine has not been defined.

Table 8.1

9. 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 9 for details regarding ASCII commands.

9.1. Standalone Command Set

Command	R/W	Description	Example
;	-	Comment notation. Comments out any text following ; in the same line.	;This is a comment
ABORT	W	Immediately stop all motion for all axis.	ABORT
ABORT[axis]	W	Immediately stop all motion for a single axis	ABORTX ABORTZ
ABS	W	Set the move mode to absolute mode.	ABS X1000 ;move to position 1000

ACC	R/W	Set/get the global acceleration setting. Unit is in milliseconds.	ACC=500
			ACC=V1

ACC[axis]	R/W	Set/get the individual acceleration setting. Unit is in milliseconds.	ACCX=500
			ACCY=V1

DELAY	W	Set a delay in milliseconds. Assigned value is a 32-bit unsigned integer or a variable.	DELAY=1000 ;1 second
			DELAY=V1 ;assign to variable

DI	R	Return status of digital inputs. See Table 6.3 for bitwise assignment.	IF DI=0
			  DO=1 ;Turn on DO1
			ENDIF
			V2=DI

DI[1-4]	R	Get individual bit status of digital inputs. Will return [0,1]. See Table 6.3 for bitwise assignment.	IF DI1=0
			  DO=1 ;Turn on DO1
			ENDIF
			V3=DI1

DO	R/W	Set/get digital output status. See Table 6.4 for bitwise assignment.	DO=2 ;Turn on DO2
DO[1-4]	R/W	Set/get individual bit status of digital outputs. Range for the bit assigned digital outputs is [0,1].	DO2=1 ;Turn on DO2
DN		Return the device name.	[4CX00-4CX99]
DN=[value]		Set the device name.	OK
EO	R/W	Set/get the enable output status. Refer to Table 6.1	EO=3 ;Enable the X and Y motor
EO[1-4]	R/W	Set/get individual bit status of the enable outputs.	EO3=1 ;Enable the Z motor
GOSUB [0-31]	-	Call a subroutine that has been previously stored to flash memory.	GOSUB 0 END
HLHOME[axis][+/-]	W	Home the motor using the home input at low and high speeds in the specified direction. See section 6.8.2 for details.	HLHOMEX+ ;positive X home WAITX ;wait for X home move
HOME[axis][+/-]	W	Home the motor using the home input at high speed in specified direction. See section 6.8.1 for details.	HOMEX- ;negative X home WAITX ;wait for X home move

HSPD	R/W	Set/get the global high speed setting. Unit is in pulses/second.	HSPD=1000
			HSPD=V1

HSPD[axis]	R/W	Set/get the individual high speed setting. Unit is in pulses/second.	HSPDY=1000
			HSPDZ=V1

IF
ELSEIF
ELSE
ENDIF
	-	Perform a standard IF/ELSEIF/ELSE conditional. Any command with read ability can be used in a conditional.
		
		ENDIF should be used to close off an IF statement.
		
		Conditions [=, >, <, >=, <=, !=] are available
			IF DI1=0
			DO=1 ;Turn on DO1
			ELSEIF DI2=0
			DO=2; Turn on DO2
			ELSE
			DO=0; Turn off DO
			ENDIF

INC	W	Set the move mode to incremental mode.	INC X1000 ;increment by 1000
JOG[axis][+/-]	W	Move the motor indefinitely in the specified direction.	JOGX+
LHOME[axis][+/-]	W	Home the motor using the limit inputs in the specified directions. See section 6.8.3 for details.	LHOMEX+ ;positive home WAITX

LSPD	R/W	Set/get the global low speed setting. Unit is in pulses/second.	LSPD=100
			LSPD=V3

LSPD[axis]	R/W	Set/get the individual low speed setting. Unit is in pulses/second.	LSPDX=100
			LSPDY=V1

MIO[axis]	R	Get motor IO status of specified axis.	Refer to Table 6.0
MST[axis]	R	Get the current motor status of the motor of an axis.	Refer to Table 6.1
PRG [0-3] END	-	Used to define the beginning and end of a main program. Four standalone programs are available	PRG 0 ;main program END
PS[axis]	R	Get the current motor speed.	V2=PSX ;Set V2 to speed

P[axis]	R/W	Set/get the current motor position.	PX=1000 ;Set to X pos to 1000
			V1=PY ;Read current Y position

SR[0-3]	W	Set the standalone control for the specified program. See Table 6.16.	SR0=0 ;Turn off program 0
STOP	W	Stop all motion using a decelerated stop.	
STOP[axis]	W	Stop motion using a decelerated stop for an individual axis.	STOPX STOPY
STORE	W	Store settings to flash.	STORE

SUB [0-31]
ENDSUB
	-	Defines the beginning of a subroutine. ENDSUB should be used to define the end of the subroutine.	SUB 1
			DO=4
			ENDSUB

TOC	W	Sets the communication time-out parameter. Units is in milliseconds.	TOC=1000 ;1 second time-out
U[position]	W	If in absolute mode, move the U motor to [position]. If in incremental mode, move the motor to [current position] + [position].	U1000

V[0-63]	R/W	Set/get standalone variables.
		
		The following operations are available:
		[+] Addition
		[-] Subtraction
		[*] Multiplication
		[/] Division
		[%] Modulus
		[>>] Bit shift right
		[<<] Bit shift left
		[&] Bitwise AND
			V1=12345 ;Set V1 to 12345
			V2=V1+1;Set V2 to V1 + 1
			V3=DI ;Set V3 to DI
			V4=DO ;Set V4 to DO V5=~EO
			;Set V5 to NOT EO

		[|] Bitwise OR [~] Bitwise NOT	
WAIT[axis]	W	Wait for current motion to complete before processing the next line.	X1000 ;move to position 1000 WAITX ;wait for move

WHILE
ENDWHILE
	-	Perform a standard WHILE loop within the standalone program. ENDWHILE should be used to close off a WHILE loop.
		
		Conditions [=, >, <, >=, <=, !=] are available.
			WHILE 1=1 ;Forever loop
			  DO=1 ;Turn on DO1
			  DO=0 ;Turn off DO1
			ENDWHILE

X[position]	W	If in absolute mode, move the X motor to [position]. If in incremental mode, move the motor to [current position] + [position].	X1000
Y[position]	W	If in absolute mode, move the Y motor to [position]. If in incremental mode, move the motor to [current position] + [position].	Y1000
Z[position]	W	If in absolute mode, move the Z motor to [position]. If in incremental mode, move the motor to [current position] + [position].	Z1000

Table 9.0

9.2. Example Standalone Programs

9.2.1. Standalone Example Program 1 – Single Thread

Task: Set the high speed and low speed and move the motor to 1000 and back to 0.

```
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
X1000             ;* Move to 1000
WAITX             ;* Wait for X-axis move to complete
X0                ;* Move to zero
WAITX             ;* Wait for X-axis move to complete
END               ;* End of the program
```

9.2.2. Standalone Example Program 2 – Single Thread

Task: Move the motor back and forth indefinitely between position 1000 and 0.

```
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
```

9.2.3. Standalone Example Program 3 – Single Thread

Task: Move the motor back and forth 10 times between position 1000 and 0.

```
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
V1=0                ;* Set variable 1 to value 0
WHILE V1<10         ;* Loop while variable 1 is less than 10
        X0          ;* Move to zero
        WAITX       ;* Wait for X-axis move to complete
        X1000       ;* Move to 1000
        WAITX       ;* Wait for X-axis move to complete
        V1=V1+1     ;* Increment variable 1
ENDWHILE            ;* Go back to WHILE statement
END
```

9.2.4. Standalone Example Program 4 – Single Thread

Task: Move the motor back and forth between position 1000 and 0 only if the digital input 1 is turned on.

```
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
        IF DI1=1            ;* If digital input 1 is on, execute the statements
                X0          ;* Move to zero
                WAITX       ;* Wait for X-axis move to complete
                X1000       ;* Move to 1000
                WAITX       ;* Wait for X-axis move to complete
        ENDIF
ENDWHILE                    ;* Go back to WHILE statement
END
```

9.2.5. Standalone Example Program 5 – Single Thread

Task: Using a subroutine, increment the motor by 1000 whenever the DI1 rising edge is detected.

```
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
V1=0                        ;* Set variable 1 to zero
WHILE 1=1                   ;* Forever loop
        IF DI1=1            ;* If digital input 1 is on, execute the statements
                GOSUB 1     ;* Move to zero
        ENDIF
ENDWHILE                    ;* Go back to WHILE statement
END

SUB 1
        XV1                 ;* Move to V1 target position
        WAITX               ;* Wait for X-axis move to complete
        V1=V1+1000          ;* Increment V1 by 1000
        WHILE DI1=1         ;* Wait until the DI1 is turned off so that
        ENDWHILE            ;* multiple increment are not continuously done
ENDSUB
```

9.2.6. Standalone Example Program 6 – Single Thread

Task: If digital input 1 is on, move to position 1000. If digital input 2 is on, move to position 2000. If digital input 3 is on, move to 3000. If digital input 5 is on, home the motor in negative direction. Use digital output 1 to indicate that the motor is moving or not moving.

```
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
        IF DI1=1            ;* If digital input 1 is on
                X1000       ;* Move to 1000
                WAITX       ;* Wait for X-axis move to complete
        ELSEIF DI2=1        ;* If digital input 2 is on
                X2000       ;* Move to 2000
                WAITX       ;* Wait for X-axis move to complete
        ELSEIF DI3=1        ;* If digital input 3 is on
                X3000       ;* Move to 3000
                WAITX       ;* Wait for X-axis move to complete
        ELSEIF DI5=1        ;* If digital input 5 is on
                HOMEX-      ;* Home the motor in negative direction
                WAITX          ;* Wait for X-axis home move to complete
        ENDIF
        V1=MSTX                ;* Store the motor 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
ENDWHILE                       ;* Go back to WHILE statement
END
```

9.2.7. Standalone Example Program 7 – Multi Thread

Task: 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.

```
PRG 0                          ;* Start of Program 0
HSPD=20000                     ;* Set high speed to 20000 pulses/sec
LSPD=500                       ;* Set low speed to 500 pulses/sec
ACC=500                        ;* Set acceleration to 500 msec
WHILE 1=1                      ;* Forever loop
        X0                     ;* Move to position 0
        WAITX                  ;* Wait for the move to complete
        X1000                  ;* Move to position 1000
        WAITX                  ;* Wait for the move to complete
ENDWHILE                       ;* Go back to WHILE statement
END                            ;* End Program 0

PRG 1                          ;* Start of Program 1
WHILE 1=1                      ;* Forever loop
        IF DI1=1               ;* If digital input 1 is triggered
                ABORTX         ;* Stop movement
                SR0=0          ;* Stop Program 1
        ELSE                   ;* If digital input 1 is not triggered
                SR0=1          ;* Run Program 1
        ENDIF
ENDWHILE                       ;* Go back to WHILE statement
END                            ;* End Program 1
```

9.2.8. Standalone Example Program 8 – Multi Thread

Task: Program 0 will continuously move the motor between positions 0 and 1000. Simultaneously, program 1 will monitor the communication time-out parameter and triggers 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 the error occurs.

```
PRG 0                          ;* Start of Program 0
HSPD=1000                      ;* Set high speed to 1000 pulses/sec
LSPD=500                       ;* Set low speed to 500 pulses/sec
ACC=500                        ;* Set acceleration to 500 msec
TOC=5000                       ;* Set time-out alarm to 5 seconds
EO=1                           ;* Enable motor
WHILE 1=1                      ;* Forever loop
        X0                     ;* Move to position 0
        WAITX                  ;* Wait for the move to complete
        X1000                  ;* Move to position 1000
        WAITX                  ;* Wait for the move to complete
ENDWHILE                       ;* Go back to WHILE statement
END                            ;* End Program 0

PRG 1                          ;* Start of Program 1
WHILE 1=1                      ;* Forever loop
        V1=MSTX&2048           ;* Get bit time-out counter alarm variable
        IF V1 = 2048           ;* If time-out counter alarm is on
                SR0=0          ;* Stop program 0
                ABORTX         ;* Abort the motor
                DO=0           ;* Set DO=0
                DELAY=3000     ;* Delay 3 seconds
                SR0=1          ;* Turn program 0 back on
                DO=1           ;* Set DO=1
        ENDIF
        ENDWHILE               ;* Go back to WHILE statement
END                            ;* End Program 1
```

A: Speed Settings

HSPD value [PPS] [1]	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 - 400K	4	10	1	800	

Table A.0

[1]If StepNLoop is enabled, the [HSPD range] values needs to be transposed from PPS (pulse/sec) to EPS (encoder counts/sec) using the following formula:

EPS = PPS / Step-N-Loop Ratio

A.1. Acceleration Range

The allowable acceleration/deceleration values depend on the **LSPD** and **HSPD** settings.

The minimum acceleration/deceleration setting for a given high speed and low speed is shown in Table A.0.

The maximum acceleration/deceleration setting for a given high speed and low speed can be calculated using the formula:

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

$$\boxed{\textbf{Max ACC} = ((HS - LS) / \delta) \times 1000 \text{ [ms]}}$$

Figure A.0

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)

A.2. Acceleration Range – Positional Move

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

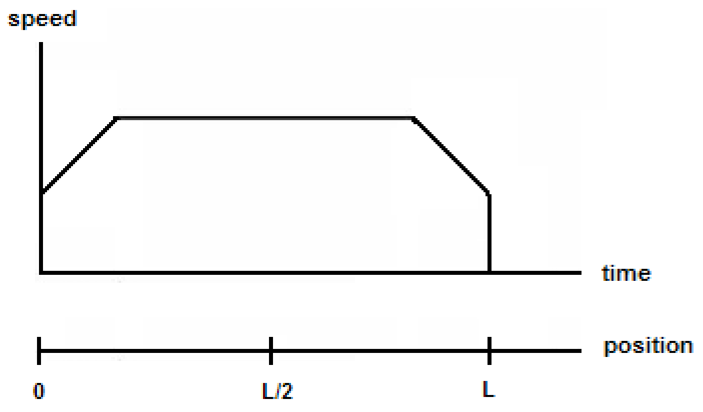


Figure A.1

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.

Contact Information

Arcus Technology, Inc.

3159 Independence Dr
Livermore, CA 94551
925-373-8800

www.arcus-technology.com