

**Performax 4CX-SA**

**4-Axis**

**Stepper Motor Controller**

**Standalone Version**

**Manual**



COPYRIGHT © 2005 ARCUS, ALL RIGHTS RESERVED

First edition, September 2005

ARCUS TECHNOLOGY copyrights this document. You may not reproduce or translate into any language in any form and means any part of this publication without the written permission from ARCUS.

ARCUS 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:**

- 1.00 – 1<sup>st</sup> Release
- 1.08 – 2<sup>nd</sup> Release
- 1.09 – 3<sup>rd</sup> Release

**Firmware Compatibility:**

†V116BL

†If your module's firmware version number is less than the listed value, contact Arcus for the appropriate documentation. Arcus reserves the right to change the firmware without notice.

## Table of Contents

|  |    |
|--|----|
| 1. Introduction                            | 6  |
| Features                                   | 6  |
| 2. Electrical and Thermal Specifications   | 7  |
| Power Requirement                          | 7  |
| Temperature Ratings †                      | 7  |
| Digital Inputs †                           | 7  |
| Digital Outputs                            | 7  |
| 3. Dimensions                              | 8  |
| 4. Pin Descriptions                        | 9  |
| 2-Pin Connector (5.08mm)                   | 9  |
| Left Side 8-Pin Connector (3.81mm)         | 9  |
| Left Side 10-Pin Connector (3.81mm)        | 10 |
| Right Side 8-Pin Connector (3.81mm)        | 10 |
| Right Side 10-Pin Connector (3.81mm)       | 11 |
| Pulse, Direction, and Enable Outputs       | 12 |
| Limits, Home, and Digital Inputs           | 12 |
| Digital Outputs                            | 13 |
| 5. Getting Started                         | 14 |
| Typical Setup                              | 14 |
| Windows GUI                                | 15 |
| Main Control Screen                        | 15 |
| 6. Motion Control Feature Overview         | 24 |
| Motion Profile                             | 24 |
| Target Motion                              | 24 |
| Homing                                     | 25 |
| Home Input Only (High speed only)          | 25 |
| Home Input Only (High speed and low speed) | 25 |
| Limit Only                                 | 26 |
| Jog Move                                   | 27 |
| Stopping Motor                             | 27 |
| Motor Position                             | 27 |
| Pulse Speed                                | 27 |
| Communication Time-out Feature (Watchdog)  | 27 |
| Motor Status                               | 27 |
| Motor IO Status                            | 27 |
| Digital Inputs                             | 28 |
| Digital Outputs                            | 28 |
| Motor Power                                | 29 |
| Limit Switch Function                      | 29 |
| Device Number                              | 29 |
| Standalone Program Specification           | 30 |
| Storing to Flash                           | 31 |
| 7. Communication                           | 32 |
| USB Communication API Functions            | 32 |
| USB Communication Issues                   | 33 |

|  |    |
|--|----|
| 8. ASCII Language Specification _____      | 34 |
| 9. Standalone Language Specification _____ | 38 |
| ;  | 38 |
| ABORT _____                                | 38 |
| ABORT[axis] _____                          | 38 |
| ABS _____                                  | 38 |
| ACC _____                                  | 39 |
| ACC[axis] _____                            | 39 |
| DELAY _____                                | 40 |
| DI _____                                   | 40 |
| DI[1-4] _____                              | 40 |
| DN _____                                   | 41 |
| DO _____                                   | 41 |
| DO[1-4] _____                              | 41 |
| ELSE _____                                 | 42 |
| ELSEIF _____                               | 42 |
| END _____                                  | 43 |
| ENDIF _____                                | 43 |
| ENDSUB _____                               | 43 |
| ENDWHILE _____                             | 43 |
| EO _____                                   | 44 |
| EO[1-4] _____                              | 44 |
| GOSUB _____                                | 45 |
| HLHOME[axis][+ or -] _____                 | 45 |
| HOME[axis][+ or -] _____                   | 45 |
| HSPD _____                                 | 45 |
| HSPD[axis] _____                           | 46 |
| IF _____                                   | 46 |
| INC _____                                  | 47 |
| JOG[axis] _____                            | 47 |
| LHOME[axis][+ or -] _____                  | 47 |
| LSPD _____                                 | 48 |
| LSPD[axis] _____                           | 48 |
| MIO[axis] _____                            | 49 |
| MST[axis] _____                            | 49 |
| P[axis] _____                              | 49 |
| PS[axis] _____                             | 50 |
| SR[0,3] _____                              | 50 |
| STOP _____                                 | 50 |
| STOP[axis] _____                           | 51 |
| STORE _____                                | 51 |
| SUB _____                                  | 51 |
| TOC _____                                  | 51 |
| U _____                                    | 52 |
| V[index] _____                             | 52 |
| WAIT[axis] _____                           | 53 |

---

|  |    |
|--|----|
| WHILE _____  | 53 |
| X _____  | 54 |
| Y _____  | 54 |
| Z _____  | 54 |
| 10. Example Standalone Programs _____              | 55 |
| Standalone Example Program 1 – Single Thread _____ | 55 |
| Standalone Example Program 2 – Single Thread _____ | 55 |
| Standalone Example Program 3 – Single Thread _____ | 55 |
| Standalone Example Program 4 – Single Thread _____ | 56 |
| Standalone Example Program 5 – Single Thread _____ | 56 |
| Standalone Example Program 6 – Single Thread _____ | 57 |
| Standalone Example Program 7 – Multi Thread _____  | 58 |
| Standalone Example Program 8 – Multi Thread _____  | 59 |

# 1. Introduction

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

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

Windows and Linux drivers, as well as sample source code, are available to aid you in your software development.

## **Features**

### **PMX-4CX-SA**

- USB 2.0 communication
- Standalone programmable
- Opto-isolated I/O
  - 4 x inputs
  - 4 x outputs
  - 4 x +Limit/-Limit/Home inputs
- Open-collector outputs
  - 4 x Pulse, Direction, Enable
- 400K pulse output rate
- Homing routines
  - Home input only (high speed)
  - Home input only (high speed + low speed)
  - Limit only
- 12-24VDC voltage input

### **Contacting Support**

For technical support contact: [support@arcus-technology.com](mailto:support@arcus-technology.com).

Or, contact your local distributor for technical support.

---

## 2. Electrical and Thermal Specifications

### ***Power Requirement***

|                            |                       |
|----------------------------|-----------------------|
| Regulated Voltage:         | <b>+12 to +24 VDC</b> |
| Recommended Current (Max): | <b>200 mA</b>         |

### ***Temperature Ratings*** †

|                        |                        |
|------------------------|------------------------|
| Operating Temperature: | <b>0°C to +85°C</b>    |
| Storage Temperature:   | <b>-55°C to +150°C</b> |

† Based on component ratings

### ***Digital Inputs*** †

|                                |                                 |
|--------------------------------|---------------------------------|
| Type:                          | <b>Opto-isolated NPN inputs</b> |
| Opto-isolator supply:          | <b>+12 to +24 VDC</b>           |
| Maximum forward diode current: | <b>90 mA</b>                    |

† Includes limits and home

### ***Digital Outputs***

|                           |   |
|---------------------------|---|
| Type:                     | <b>Opto-isolated open-collector NPN outputs</b> |
| Max voltage at collector: | <b>+24 VDC</b>                                  |
| Max sink current at 24VDC | <b>†90 mA</b>                                   |

† A current limiting resistor is required

### 3. Dimensions

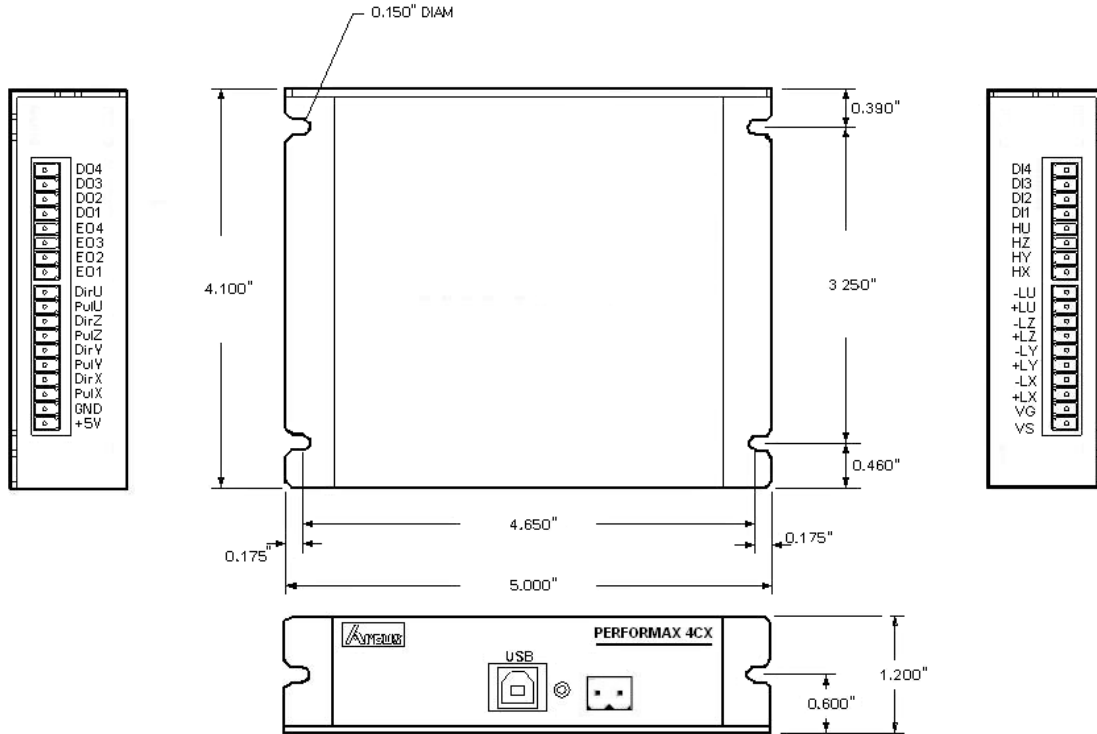


Figure 2.0

## 4. Pin Descriptions

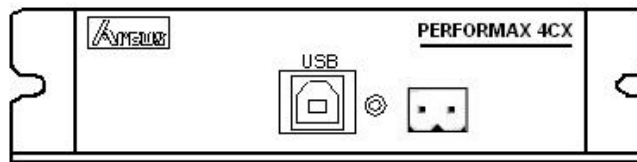


Figure 4.0

### 2-Pin Connector (5.08mm)

| Pin # | In/Out | Name | Description                |
|-------|--------|------|----------------------------|
| 1     | I      | GND  | Ground                     |
| 2     | I      | PWR  | 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: †EDZ950/2

† Other 5.08mm compatible connectors can be used.

### Left Side 8-Pin Connector (3.81mm)

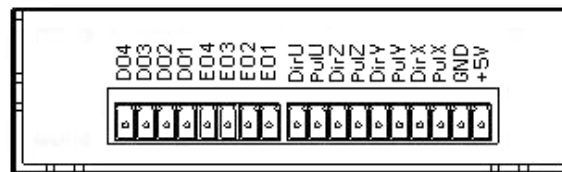


Figure 4.1

| 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 4.1

Mating Connector Description: 8 pin 0.15" (3.81mm) connector  
Mating Connector Manufacturer: On-Shore

Mating Connector Manufacturer Part: †EDZ1550/8

† Other 3.81 compatible connectors can be used.

**Left Side 10-Pin Connector (3.81mm)**

| Pin # | In/Out | Name | Description             |
|-------|--------|------|-------------------------|
| 1     | O      | DirU | U Axis Direction Output |
| 2     | O      | PulU | U Axis Pulse Output     |
| 3     | O      | DirZ | Z Axis Direction Output |
| 4     | O      | PulZ | Z Axis Pulse Output     |
| 5     | O      | DirY | Y Axis Direction Output |
| 6     | O      | PulY | Y Axis Pulse Output     |
| 7     | O      | DirX | X Axis Direction Output |
| 8     | O      | PulX | X Axis Pulse Output     |
| 9     | I      | GND  | Ground                  |
| 10    | O      | +5V  | +5 Volt Supply Output   |

Table 4.2

Mating Connector Description: 10 pin 0.15” (3.81mm) connector

Mating Connector Manufacturer: On-Shore

Mating Connector Manufacturer Part: †EDZ1550/10

† Other 3.81 compatible connectors can be used.

**Right Side 8-Pin Connector (3.81mm)**



Figure 4.2

| Pin # | In/Out | Name | Description       |
|-------|--------|------|-------------------|
| 1     | I      | HX   | X Axis Home Input |
| 2     | I      | HY   | Y Axis Home Input |
| 3     | I      | HZ   | Z Axis Home Input |
| 4     | I      | HU   | U Axis Home Input |
| 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 4.3

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

† Other 3.81 compatible connectors can be used.

***Right Side 10-Pin Connector (3.81mm)***

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

Table 4.4

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

† Other 3.81 compatible connectors can be used.

### ***Pulse, Direction, and Enable Outputs***

The pulse, direction, and enable outputs are open collector outputs using 74LS07.

Figure 4.3 shows an example of the pulse, direction, and enable connections to a driver. Maximum sink current is 40mA.

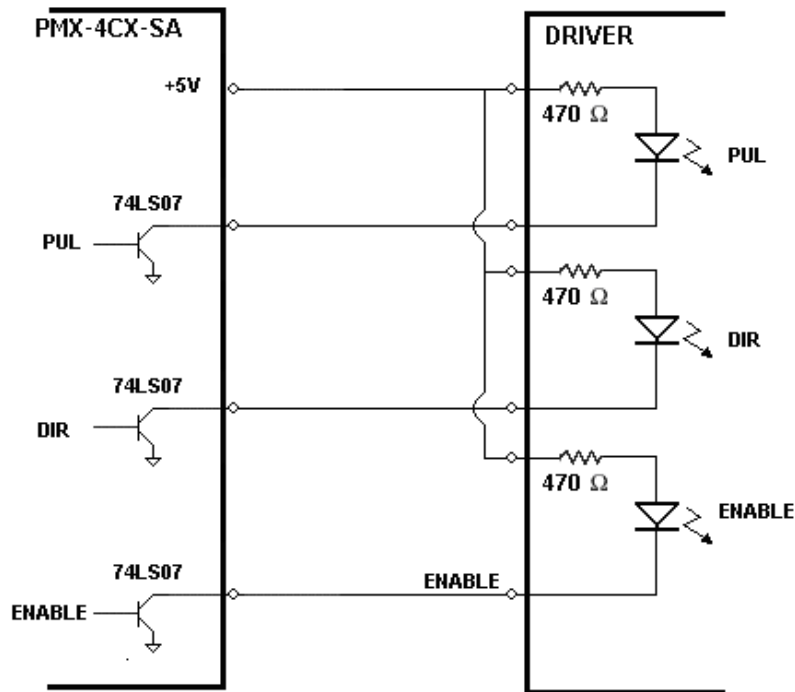


Figure 4.3

### ***Limits, Home, and Digital Inputs***

Figure 4.4 shows an example wiring of the home, limit and digital inputs.

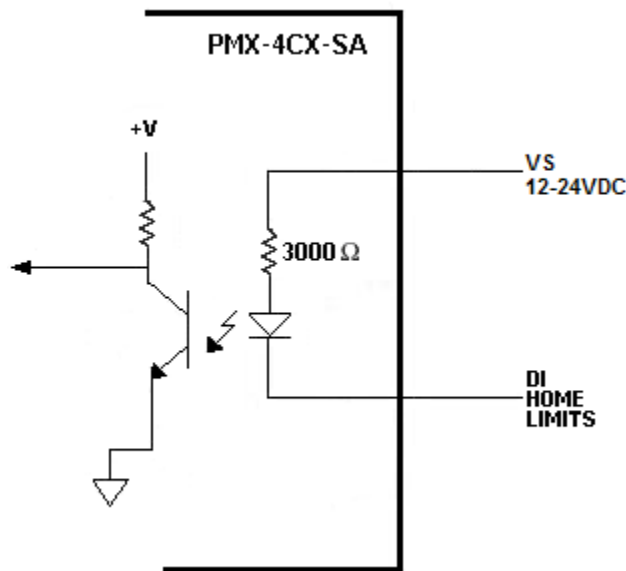


Figure 4.4

To activate an input, sink the line to opto-ground.

### **Digital Outputs**

Figure 4.5 shows an example wiring of the digital outputs.

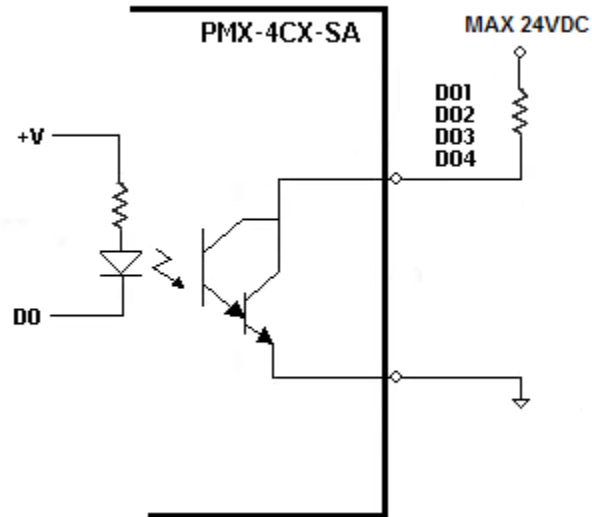


Figure 4.5

Digital outputs can sink up to 90mA of current.

## 5. Getting Started

### Typical Setup

#### PC-Controlled

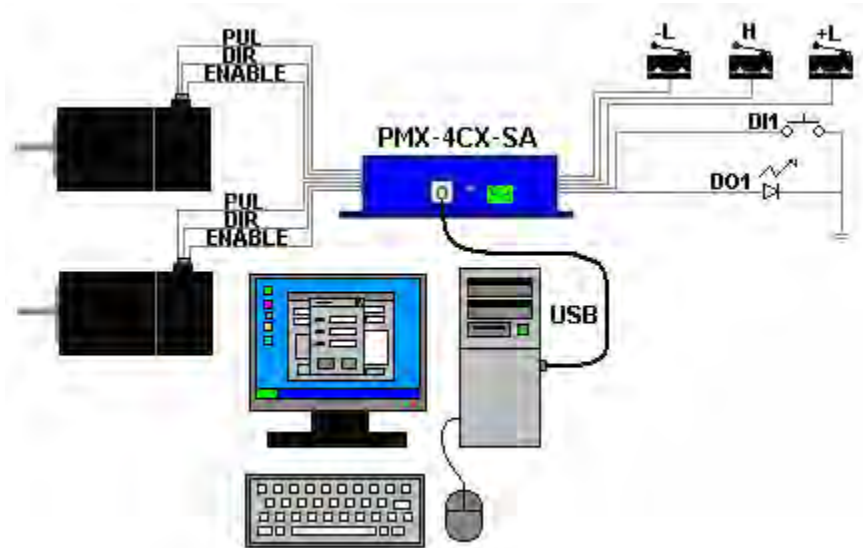


Figure 5.0

#### Stand-Alone Operation

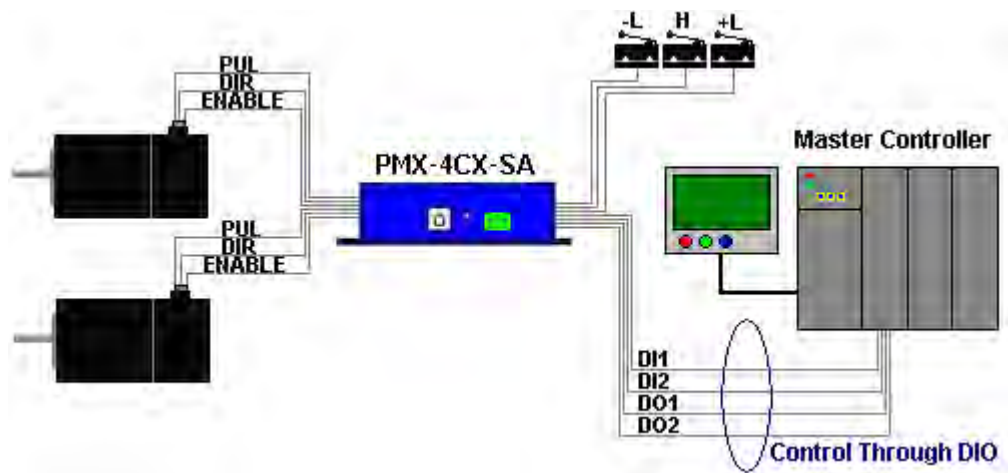


Figure 5.1

## Windows GUI

PMX-4CX-SA comes with a Windows GUI program to test, program, compile, download, and debug the controller.

Make sure that the USB driver is installed properly before running the controller.

Startup the PMX-4CX-SA GUI program and you will see following screen:

### Main Control Screen

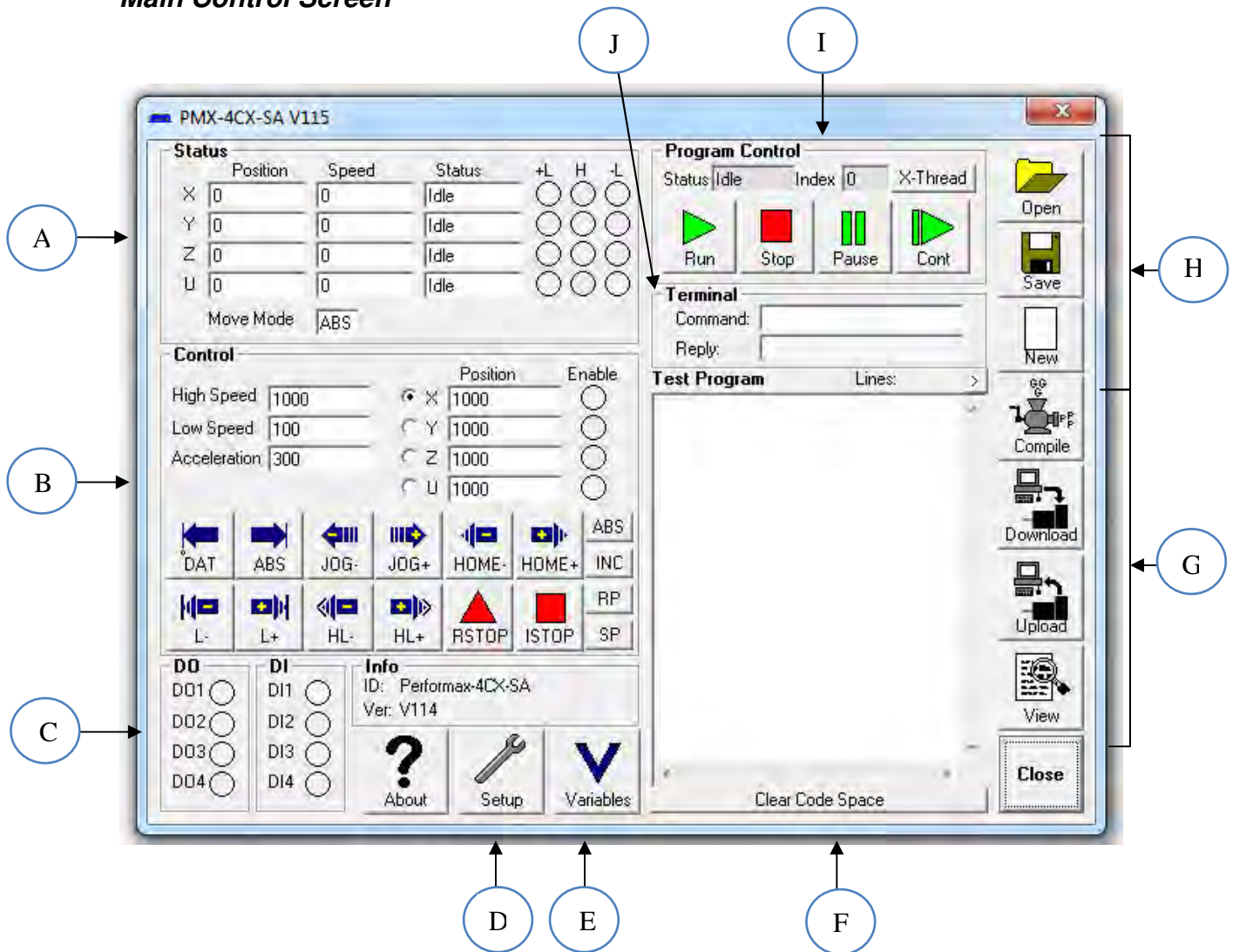


Figure 5.2

## A. Status

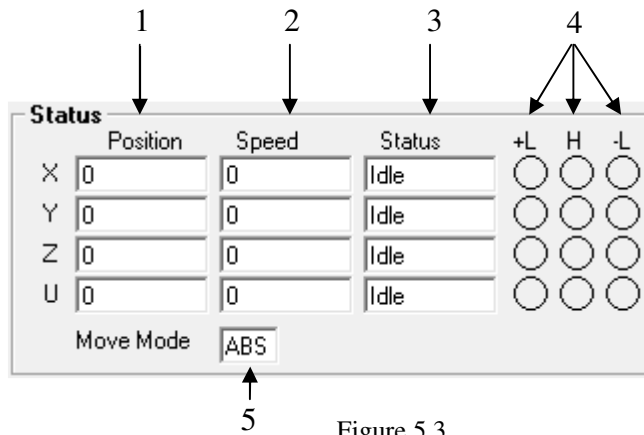


Figure 5.3

1. Current pulse position (X/Y/Z/U axis).
2. Current pulse speed (X/Y/Z/U axis).
3. Motor status (X/Y/Z/U axis).
  - a. Idle – Motor is not moving.
  - b. Accel – Motor is accelerating.
  - c. Const – Motor is moving at constant speed.
  - d. Decel – Motor is decelerating.
4. –Limit, +Limit, Home input status (X/Y/Z/U status).
5. Move mode.
  - a. ABS: On target movement, motor will move to target position.
  - b. INC: On target movement motor will increment by the target position.

## B. Control

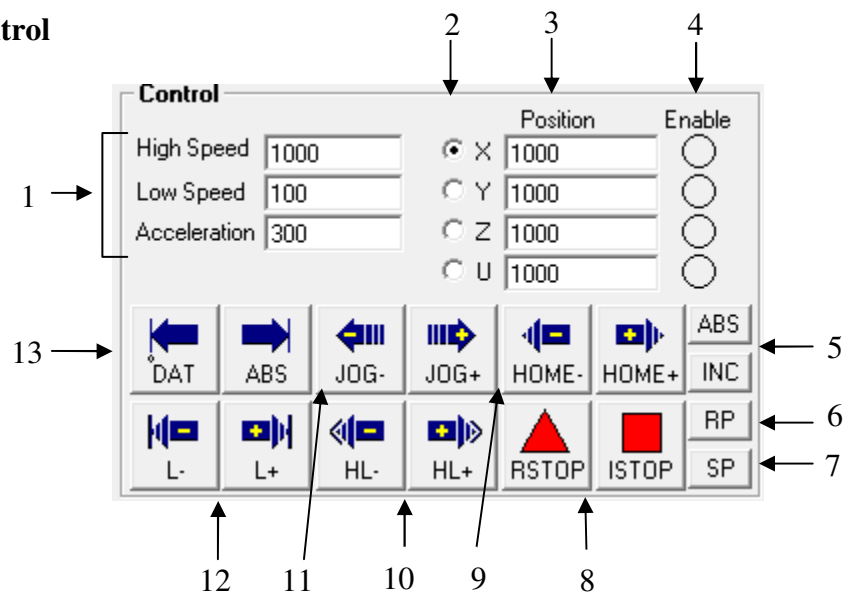


Figure 5.4

1. Global high speed, low speed, and acceleration entered here. To give each axis individual speed parameters, enter **HSPD[axis]**, **LSPD[axis]**, and **ACC[axis]** into the command line in the “Terminal” section.
2. Select the axis to control. The axis must also be enabled for the motor to respond.
3. Target position used for absolute and incremental movement.
4. Enable the driver for the indicated axis.
5. **ABS/INC** - Set the move mode to absolute or incremental for target movement.
6. **RP** - Reset the indicated motor to position zero.
7. **SP** - Set the indicated motor to the position specified in the Position field.
8. **RSTOP/ISTOP** - Stop the specified axis.
  - a. **RSTOP** stops the indicated motor with the deceleration.
  - b. **ISTOP** stops the indicated immediately, without deceleration.
9. **HOME+/HOME-** Homes the motor in the positive or negative direction.
10. **HL+/HL-** Homes the motor using both low and high speed in the positive or negative direction.
11. **JOG+/JOG-** Jogs the indicated motor in the positive or negative direction.
12. **L+/L-** Homes the motor using the Limit inputs in the positive or negative direction.
13. **ABS/DAT**:
  - a. **ABS** - Perform target movement using the Target Position field.
  - b. **DAT** – Move the motor back to the position zero.

### C. Outputs and Inputs

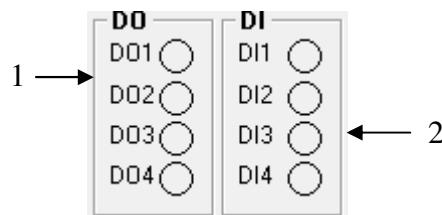


Figure 5.5

1. Set digital output status for DO1-DO4.
2. Digital input status for DI1-DI4.

## D. Setup

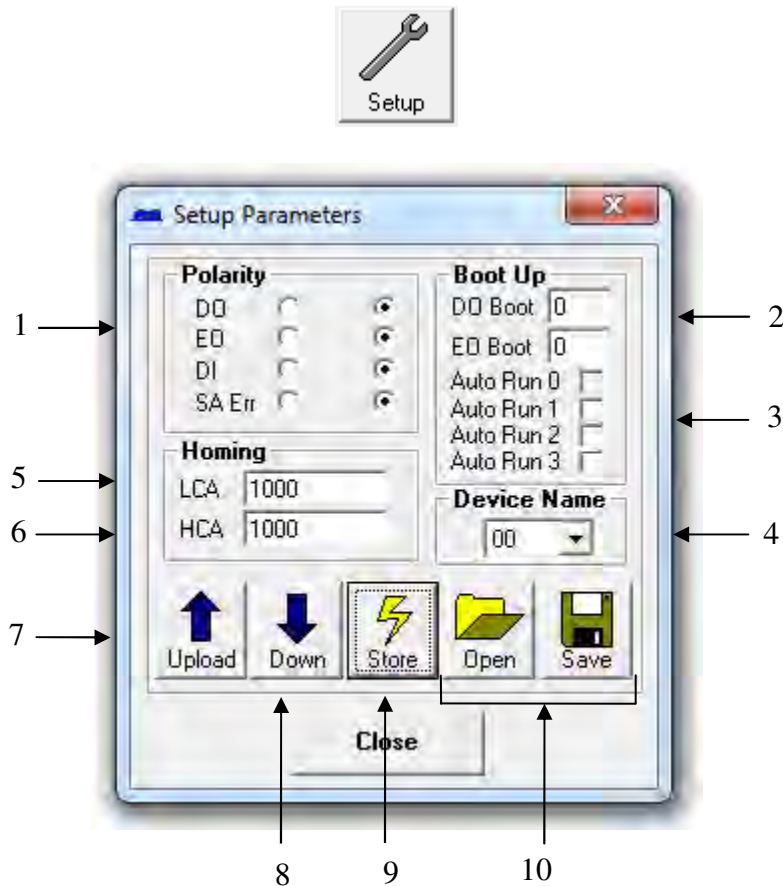


Figure 5.6

1. **Polarity** - Change the polarity of the input and outputs.
2. **DO Boot/EO Boot** – Change the boot-up state of the digital and enable outputs.
3. **Auto Run** – Have the specified standalone program run on boot-up
4. **Device Name** – Set the device name [4CX00-4CX99].
5. **LCA** – The limit correction amount is used during the limit homing routine. See the homing specification in Section 6 for details.
6. **HCA** – The home correction amount is used during various homing routines. See the homing specification in Section 6 for details.
7. **Upload** – Upload the above parameters from memory.
8. **Down** – Download the above parameters to memory. Note that a store must be executed if these values are to be preserved after a power cycle.
9. **Store** – Store the above parameters to flash memory so that they can be preserved after a power cycle.
10. **Open/Save** – Parameters can be saved to a file and read from a file.

## E. Variables

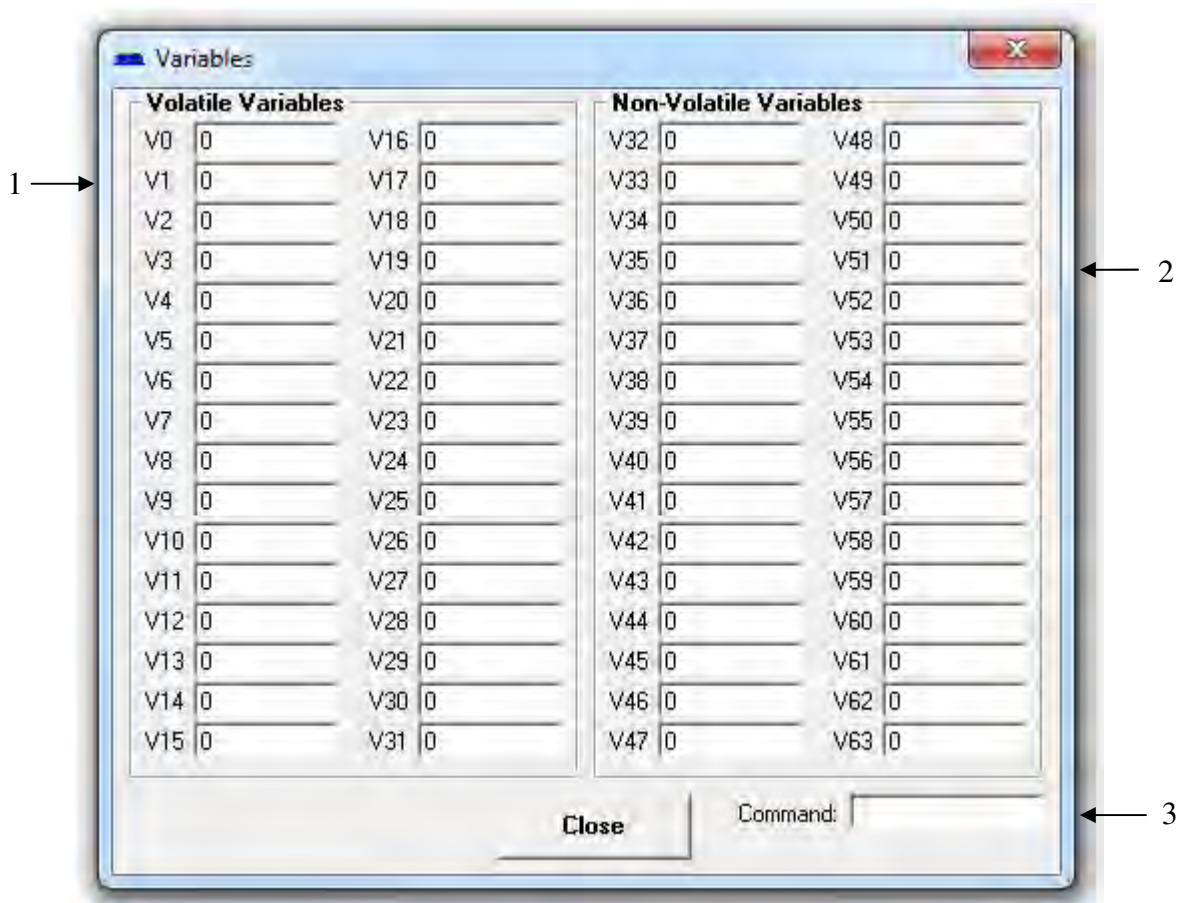


Figure 5.7

1. Volatile variables are not saved to flash and are reset to zero after a power cycle (V0-V31).
2. Non-volatile variables (V32-V63) can be saved to flash. Perform the **STORE** command to save these variables to flash.
3. Send commands to the PMX-4CX-SA through this terminal.

## F. Text Programming Box

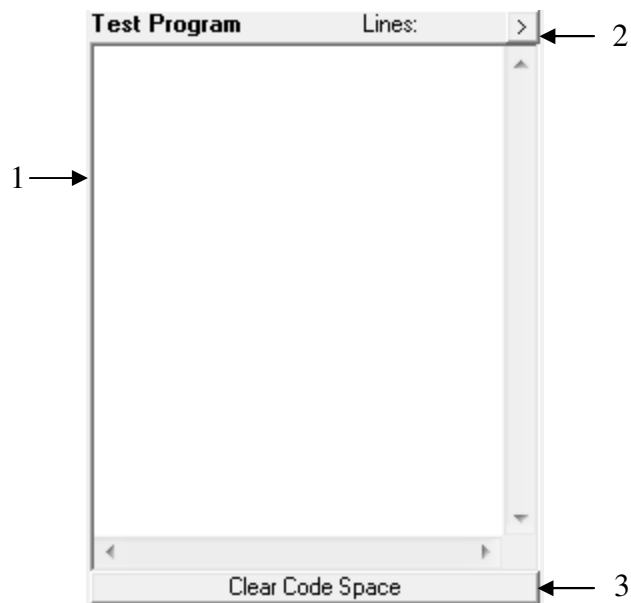


Figure 5.8

1. Text box for standalone program. See details on programming language.
2. Opens a larger Program Editor window for easier programming.
3. Clear the standalone code space on the PMX-4CX-SA

## G. Compiler

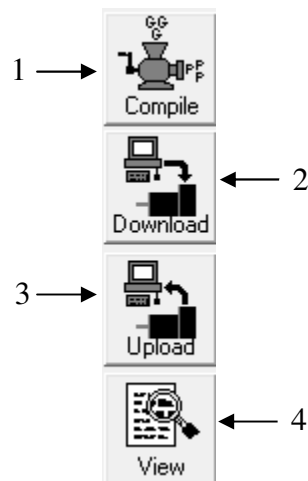


Figure 5.9

1. Compile code in the text programming box into assembly level code that the PMX-4CX-SA can understand.
2. Download the compiled code into memory. Note that the text based code must first be compiled before download.

3. Upload the standalone code that is currently on your PMX-4CX-SA. This automatically translates assembly level language into readable text-based code.
4. View compiled code for easy cutting and pasting.

## H. Program File Control

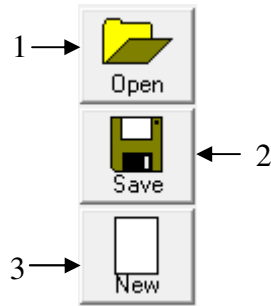


Figure 5.10

1. Open – Standalone program is loaded to the text programming box. When this button is pressed, a typical Windows file open dialog box will appear.

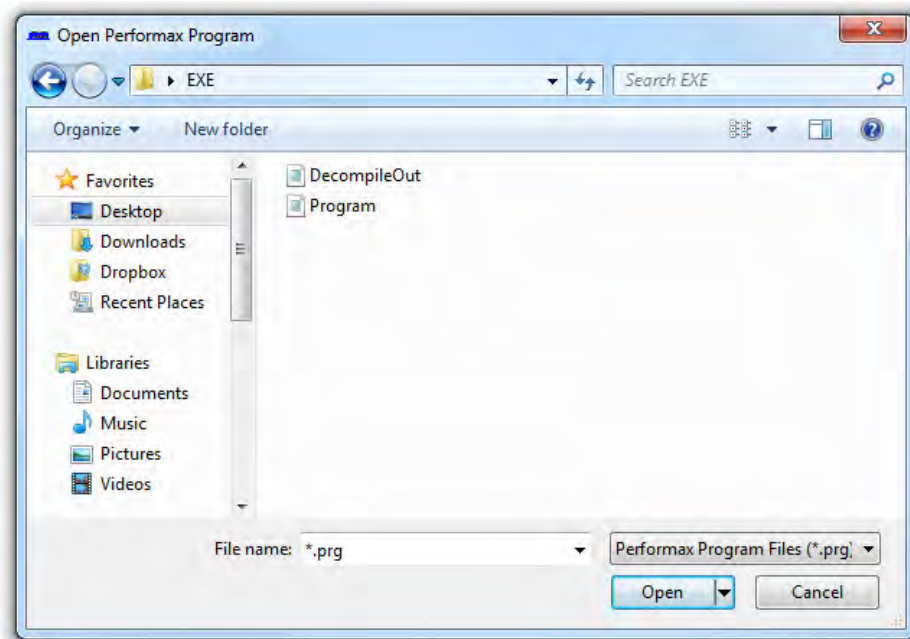


Figure 5.11

2. Save – Standalone program in the text programming box is saved to a file. When this button is pressed, a typical Windows file save dialog box will appear.

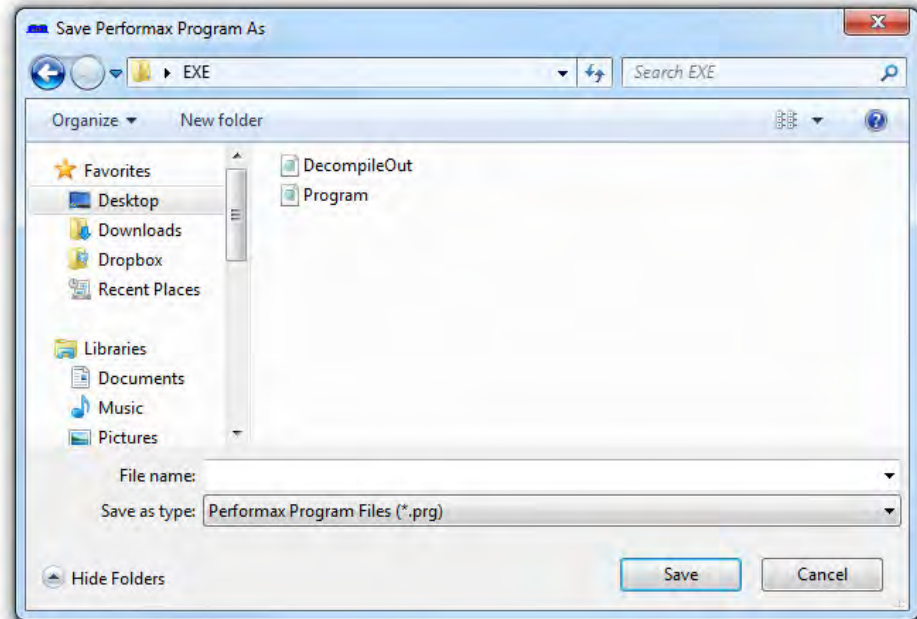


Figure 5.12

3. New – When this button is pressed, the text programming box is cleared.

### I. Program Control

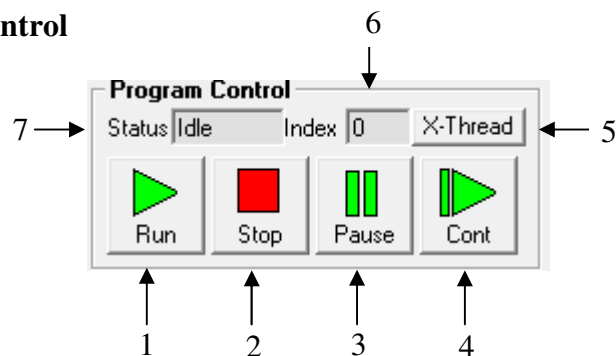


Figure 5.13

1. Run – Program is run.
2. Stop – Program is stopped.
3. Pause – Program that is running can be paused.
4. Cont – Program that is paused can be continued.
5. XThread - Open the Standalone Program Control for all standalone programs.
6. Index – Current line of low level code that is being executed
7. Status of standalone program
  - a. Idle – Program not running.
  - b. Running – Program is running.
  - c. Paused – Program is paused.
  - d. Error – Program is in error state.

## J. Command Reply

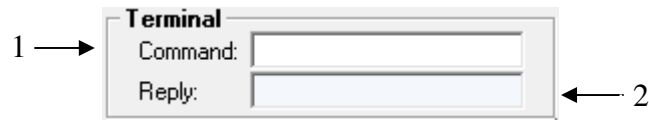


Figure 5.14

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

## 6. Motion Control Feature Overview

**Note:** All the commands described in this section are interactive commands only. Refer to the “Standalone Language Specification” section for the stand-alone code API.

### ***Motion Profile***

PMX-4CX-SA incorporates trapezoidal velocity profile as shown in figure 6.0.

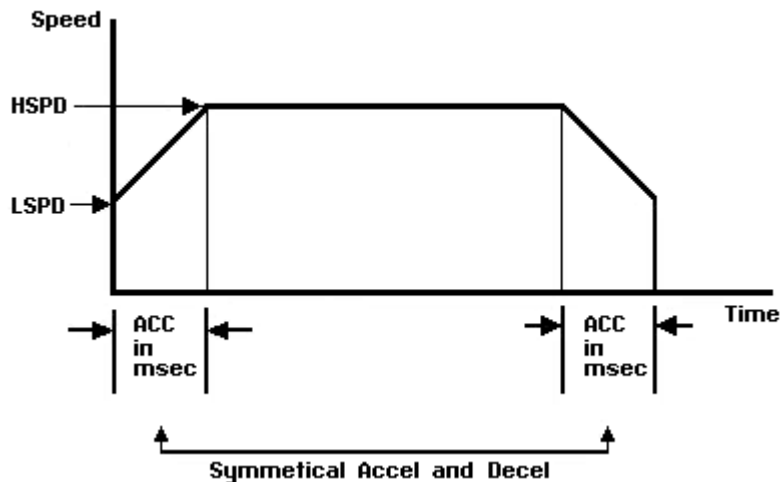


Figure 6.0

High speed and low speed are in pps(pulses/second). Use the **HSPD** and **LSPD** commands to set and get the global high speed and low speed settings. For setting and retrieving the high speed the low speed for an individual axis use the **HSPD[axis]** and **LSPD[axis]** commands.

Acceleration and deceleration time is in milliseconds and are symmetrical. Use the **ACC** command to set and retrieve the global acceleration/deceleration value. To access the acceleration/deceleration value for an individual axis use the **ACC[axis]** command.

By default, all moves use the global speed settings unless ALL parameters (i.e. high speed, low speed, and acceleration) for a certain axis are non-zero.

### ***Target Motion***

Target move, also known as absolute move, is used to move the motor to the desired position from the current position. Use the **X/Y/Z/U** command to move the axis to the target position.

PMX-4CX-SA can operate in either incremental or absolute move modes. Use the **INC** and **ABS** commands to set the move mode. Use the **MM** command to read the current move mode.

## Homing

Home search sequence involves moving the motor towards the home or limit switches and then stopping when the relevant input is detected. The PMX-4CX-SA has three different homing routines:

### Home Input Only (High speed only)

Use the **HOME[axis]+/HOME[axis]-** command. The following sequence shows the homing routine.

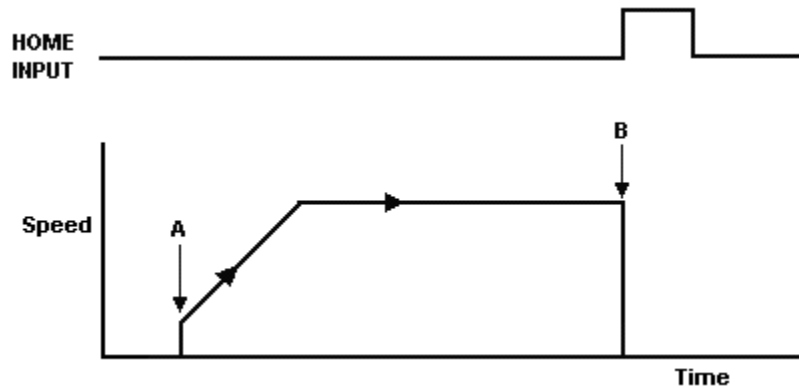


Figure 6.1

- A. Issuing home command 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 to zero and the motor stops immediately. If the home switch is triggered in the middle of the acceleration, the motor stops immediately.

### Home Input Only (High speed and low speed)

Use the **HLHOME[axis]+/HLHOME[axis]-** command. The following sequence shows the homing routine.

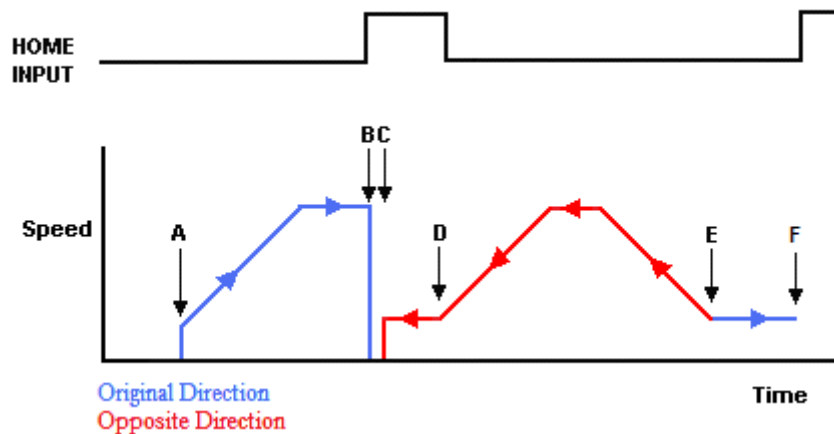


Figure 6.2

- A. Issuing low speed home command 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 to zero and the motor immediately stops immediately.
- C. Motor is begins to back out of the home switch.
- D. The motor is now off the home switch. It will continue past the home switch by the amount defined by the home correction amount (**HCA**). The direction of this movement will be the opposite of the original home command.
- E. The motor is now past the home input by the amount defined by the home correction amount (**HCA**). The motor now moves back towards the home switch at low speed.
- F. The home input 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.

### Limit Only

Use the **LHOME+/LHOME-** command. The following sequence shows the homing routine.

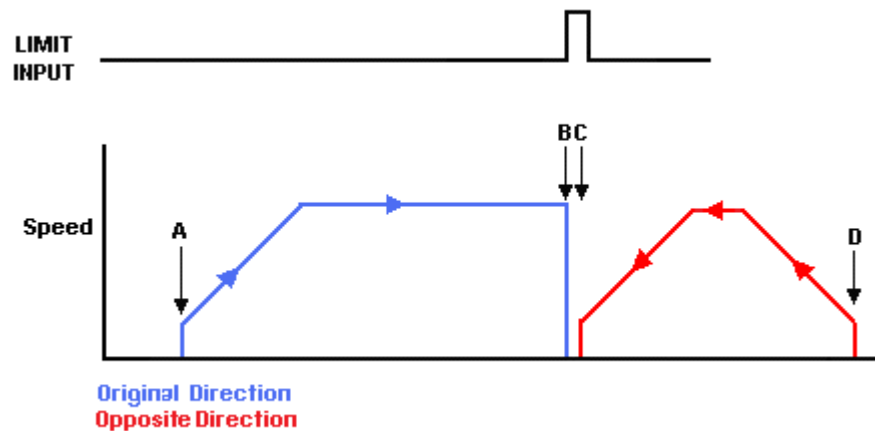


Figure 6.3

- A. Issuing a limit home command starts the motor from low speed and accelerates to high speed.
- B. As soon as the relevant limit input is triggered, the motor stops immediately.
- C. The motor position is set to the limit correction amount (**LCA**) and the motor immediately reverses direction and returns to the zero position.
- 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.

To trigger a home or limit input switch, supply the opto-supply voltage with 12-24VDC and connect the home or limit input signal to opto-supply ground.

### **Jog Move**

Jogging is available for continuous speed operation. Use the **JOG[axis]+** and **JOG[axis]-** commands to jog in the positive or negative direction for the respective axis.

### **Stopping Motor**

When motor is moving, jogging, or homing, **ABORT[axis]** command will immediately stop the indicated motor. The **ABORT** command will stop all the axes immediately. **STOP[axis]** command will decelerate the axis to low speed and then stop. The **STOP** command will stop all the axes in motion with deceleration. It is recommended to use deceleration when stopping so that there is less impact to the system.

### **Motor Position**

Motor position can be set and read using the **P[axis]** command. For example to set the X axis position to 2000, use the **PX=2000** command. To read the X axis current position use the **PX** command and the reply will contain the current position counter.

### **Pulse Speed**

Current actual pulse rate or speed can be read using the **S[axis]** command.

### **Communication Time-out Feature (Watchdog)**

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

### **Motor Status**

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

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

Table 6.0

### **Motor IO Status**

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

| Bit | Description              |
|-----|--------------------------|
| 0   | Minus Limit Input Status |
| 1   | Plus Limit Input Status  |
| 2   | Home Input Status        |

Table 6.1

### Digital Inputs

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

Read digital input status using the **DI** command. See description below:

Digital input values can also be referenced one bit at a time by the **DI[1-4]** commands. Note that the indexes are 1-based for the bit references (i.e. DI1 refers to bit 0, not bit 1)

| 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 6.2

### Digital Outputs

PMX-4CX-SA module comes with 4 digital outputs. Read and set the digital output status using the **DO** command. See description below:

| 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 6.3

Digital output values can also be referenced one bit at a time by the **DO[1-4]** commands. Note that the indexes are 1-based for the bit references (i.e. DO1 refers to bit 0, not bit 1).

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

## Motor Power

Using the **EO** command, the motor power can be enabled or disabled. See description below:

| 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 6.4

If the enable function is not used, the enable output can be used as a general purpose output. The enable output does not affect the move command. 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).

The initial state of the enable output 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.

## Limit Switch Function

If the positive limit switch is triggered while moving in positive direction, the motor will immediately stop. The same applies for the negative limit when the motor is moving in the negative direction.

To read the limit switch input status, use the **MIO[axis]** command.

## Polarity

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

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

Table 6.5

## Device Number

PMX-4CX-SA module provides the user with the ability to set the device number of a specific device. In order to make these changes, first store the desired number using the **DN** command. Note that this value must be within the range [4CX01,4CX99].

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

By default: Device name is set to: **4CX00**

### ***Standalone Program Specification***

Standalone Program Specification:

Memory size: 1785 assembly lines ~ 10.5 KB.

Note: Each line of pre-compiled code equates to 1-4 lines of assembly lines.

WAIT Statement: When writing a standalone program, it is generally necessary to wait until a motion is completed before moving on to the next line. In order to do this, the WAIT statement must be used. See the examples below:

In the example below, the variable V1 will be set immediately after the X10000 move command begins; it will not wait until the controller is idle.

```
X10000          ;* Move to position 0
V1=100
```

Conversely, in the example below, the variable V1 will not be set until the motion has been completed. V1 will only be set once the controller is idle.

```
X10000          ;* Move to position 0
WAITX          ;* Wait for the move to complete
V1=100
```

Multithreading Operation: The PMX-4CX-SA supports the simultaneous execution of up to 4 standalone programs. The **SR[0-3]** command is used to control the operation of the respective standalone program.

**Note:** Sub-routines can be shared by different threads.

Error Handling: If an error occurs during standalone execution, the program automatically jumps to SUB 31. If SUB 31 is NOT defined, the program will cease execution and go to error state. If SUB 31 is defined by the user, the code within SUB 31 is first executed, and then standalone execution continues. The return jump line will be determined by bit 3 of the **POL** register. See Table 6.5 for details.

Calling subroutines over communication: Once a subroutine is written into the flash, they can be called via USB communication using the **GS** command. The subroutines are referenced by their subroutine number [0-31]. If a subroutine number is not defined, the PMX-4CX-SA will return with an error.

Standalone Run on Boot-Up: Standalone can be configured to run on boot-up using the **SLOAD** command. See description below:

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

Table 6.6

### **Storing to Flash**

The following items are stored to flash:

| ASCII Command | Description  |
|---------------|--|
| DN            | Device Name  |
| POL           | Polarity Settings  |
| SLOAD         | Standalone program run on boot-up parameter                        |
| 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.                    |
| TOC           | Time-out counter reset value                                       |
| V32-V63       | Standalone variables. Note that on boot-up, V0-V31 are reset to 0. |

Table 6.7

*Note: Standalone program is automatically stored to flash when it is downloaded.*

## 7. Communication

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

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

Typical communication transaction time between PC and PMX-4CX-SA for sending a command from a PC and getting a reply from PMX-4CX-SA using the **fnPerformaxComSendRecv()** API function is in single digit milliseconds. This value will vary with CPU speed of PC and the type of command.

### **USB Communication API Functions**

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

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

- This function is used to get total number 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 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 the 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 timeouts. 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 a command and receive a reply. The number of bytes to read and write must be 64 characters.

### **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 flush the USB buffers of the PC and the USB device right after the communication handle is opened. 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 photo below:



Figure 7.0

## 8. ASCII Language Specification

**Note:** All the commands described in this section are interactive commands only. Refer to the “Standalone Language Specification” section for the stand-alone code API.

PMX-4CX-SA language is case sensitive. All commands should be in capital letters. An invalid command is returned “?”. Always check for proper reply when a command is sent.

| Command  | Description   | Return                    |
|--|---|---------------------------|
| ABORT  | Immediately stops all the axes in motion.                                       | <b>OK</b>                 |
| ABORTX<br>ABORTY<br>ABORTZ<br>ABORTU                         | Immediately stops the axis if in motion. For decelerate stop, use STOP command. | <b>OK</b>                 |
| ABS  | Set the move mode to absolute.  | <b>OK</b>                 |
| ACC  | Return the current global acceleration [1-1000ms]                               | <b>ms</b>                 |
| ACCX<br>ACCY<br>ACCZ<br>ACCU                                 | Return the current acceleration value for the indicated axis [1-1000ms]         | <b>ms</b>                 |
| ACC=[Value]  | Sets the global acceleration value [1-1000ms]                                   | <b>OK</b>                 |
| ACCX=[Value]<br>ACCY=[Value]<br>ACCZ=[Value]<br>ACCU=[Value] | Sets acceleration value for the indicated axis [1-1000ms]                       | <b>OK</b>                 |
| DI   | Return the status of the digital inputs   | <b>Refer to Table 6.2</b> |
| DI[1-4]  | Return the individual status of the digital inputs                              | <b>Refer to Table 6.2</b> |
| DN   | Get the device number.  | <b>[4CX00-4CX99]</b>      |
| DN=[Value]   | Set the device number [4CX00-4CX99].  | <b>OK</b>                 |
| DO   | Return the status of the digital outputs  | <b>Refer to Table 6.3</b> |
| DO[1-4]  | Return the individual status of the digital outputs                             | <b>Refer to Table 6.3</b> |
| DO=[4 bit Value]   | Set the status of the enable outputs [0-15]                                     | <b>OK</b>                 |
| DO[1-4] = [Value]  | Set the individual status of the enable outputs                                 | <b>OK</b>                 |
| DOBOOT   | Returns the Digital Output configuration parameter.                             | <b>[0-15]</b>             |
| DOBOOT=[Value]   | Set the Digital Output configuration parameter [0-15].                          | <b>OK</b>                 |
| EO   | Return the status of the enable outputs   | <b>Refer to Table 6.4</b> |
| EO[1-4]  | Return the individual status of the enable outputs                              | <b>Refer to Table 6.4</b> |
| EO=[Value]   | Set the status of the enable outputs [0-15]                                     | <b>OK</b>                 |
| EO[1-4]=[Value]  | Set the individual status of the enable outputs                                 | <b>OK</b>                 |
| EOBOOT   | Return the Enable Output configuration parameter.                               | <b>[0-15]</b>             |
| EOBOOT=[Value]   | Set the Enable Output configuration parameter on boot-up.                       | <b>OK</b>                 |
| GS[0-31]   | Call a subroutine that has been previously stored to flash memory               | <b>OK</b>                 |
| HCA  | Return the global home correction amount.                                       | <b>32-bit number</b>      |

|  |   |                         |
|--|---|-------------------------|
| HCAX<br>HCAY<br>HCAZ<br>HCAU                                     | Return the home correction amount for the indicated axis                | <b>32-bit number</b>    |
| HCA=[Value]  | Set the global home correction amount                                   | <b>OK</b>               |
| HCAX=[Value]<br>HCAY=[Value]<br>HCAZ=[Value]<br>HCAU=[Value]     | Set the home correction amount for the indicated axis                   | <b>OK</b>               |
| HLHOMEX+<br>HLHOMEY+<br>HLHOMEZ+<br>HLHOMEU+                     | Home the indicated axis at low and high speed in the positive direction | <b>OK</b>               |
| HLHOMEX-<br>HLHOMEY-<br>HLHOMEZ-<br>HLHOMEU-                     | Home the indicated axis at low and high speed in the negative direction | <b>OK</b>               |
| HOMEX+<br>HOMEY+<br>HOMEZ+<br>HOMEU+                             | Home the indicated axis in the positive direction                       | <b>OK</b>               |
| HOMEX-<br>HOMEY-<br>HOMEZ-<br>HOMEU-                             | Home the indicated axis in the negative direction                       | <b>OK</b>               |
| HSPD   | Return the global high speed setting                                    | <b>[1-400K] PPS</b>     |
| HSPDX<br>HSPDY<br>HSPDZ<br>HSPDU                                 | Return the high speed setting for the indicated axis                    | <b>[1-400K] PPS</b>     |
| HSPD=[Value]   | Set the global high speed value [1-400K]                                | <b>OK</b>               |
| HSPDX=[Value]<br>HSPDY=[Value]<br>HSPDZ=[Value]<br>HSPDU=[Value] | Set the high speed value for the indicated motor [1-400K]               | <b>OK</b>               |
| ID   | Return the product ID.  | <b>Performax-4CX-SA</b> |
| INC  | Set the move mode to incremental.                                       | <b>OK</b>               |
| JOGX+<br>JOGY+<br>JOGZ+<br>JOGU+                                 | Jog the indicated axis in positive direction                            | <b>OK</b>               |
| JOGX-<br>JOGY-<br>JOGZ-<br>JOGU-                                 | Jog the indicated axis in negative direction                            | <b>OK</b>               |
| LCA  | Returns the global limit correction amount.                             | <b>32-bit number</b>    |
| LCAX<br>LCAY<br>LCAZ<br>LCAU                                     | Return the limit correction amount for the indicated axis               | <b>32-bit number</b>    |
| LCA=[Value]  | Set the global limit correction amount                                  | <b>OK</b>               |
| LCAX=[Value]<br>LCAY=[Value]<br>LCAZ=[Value]                     | Set the limit correction amount for the indicated axis                  | <b>OK</b>               |

|  |   |                           |
|--|---|---------------------------|
| LCAU=[Value]   |   |                           |
| LHOMEX+<br>LHOMEY+<br>LHOMEZ+<br>LHOMEU+                         | Home the indicated axis using the Limit inputs in the positive direction  | <b>OK</b>                 |
| LHOMEX-<br>LHOMEY-<br>LHOMEZ-<br>LHOMEU-                         | Home the indicated axis using the Limit inputs in the negative direction  | <b>OK</b>                 |
| LSPD   | Return the global low speed setting   | <b>[1-400K] PPS</b>       |
| LSPDX<br>LSPDY<br>LSPDZ<br>LSPDU                                 | Return the low speed setting for the indicated axis   | <b>[1-400K] PPS</b>       |
| LSPD=[Value]   | Set the global low speed value [1-400K]   | <b>OK</b>                 |
| LSPDX=[Value]<br>LSPDY=[Value]<br>LSPDZ=[Value]<br>LSPDU=[Value] | Set the low speed value for the indicated axis [1-400K]   | <b>OK</b>                 |
| MIOX<br>MIOY<br>MIOZ<br>MIOU                                     | Return the motor IO status  | <b>Refer to Table 6.1</b> |
| MSTX<br>MSTY<br>MSTZ<br>MSTU                                     | Return the motor status   | <b>Refer to Table 6.0</b> |
| POL  | Return the current polarity   | <b>Refer to Table 6.5</b> |
| POL=[Value]  | Set the polarity. Refer to Table 6.5  | <b>OK</b>                 |
| PX<br>PY<br>PZ<br>PU   | Returns current position value for the indicated axis.  | <b>32-bit number</b>      |
| PX=[Value]<br>PY=[Value]<br>PZ=[Value]<br>PU=[Value]             | Sets the current position value for the indicated axis.   | <b>OK</b>                 |
| SA[0-1784]   | Return the indicated standalone line  |                           |
| SA[0-1784]=[Value]   | Set the indicated standalone line   | <b>OK</b>                 |
| SASTAT[Index]  | Return the specified standalone program status<br>0 – Stopped<br>1 – Running<br>2 – Paused<br>3 – In Error  | <b>[0-3]</b>              |
| SLOAD  | Return the RunOnBoot parameter  | <b>Refer to Table 6.6</b> |
| SLOAD=[Value]  | Set the RunOnBoot parameter. Refer to table 6.6   | <b>OK</b>                 |
| SPC[0-3]   | Return the current program counter of the indicated standalone program  | <b>[0-1784]</b>           |
| SR[0-3]=[Value]  | Control the specified standalone program:<br>0 – Stop standalone program<br>1 – Run standalone program<br>2 – Pause standalone program<br>3 – Continue standalone program | <b>OK</b>                 |
| STOP   | Stop all motion using deceleration  | <b>OK</b>                 |

|  |   |                      |
|--|---|----------------------|
| STOPX<br>STOPY<br>STOPZ<br>STOPU             | Stop the indicated axis, if in motion, using deceleration               | <b>OK</b>            |
| STORE  | Store the setting to flash. Refer to table 6.7                          | <b>OK</b>            |
| SX<br>SY<br>SZ<br>SU                         | Returns current pulse speed   |                      |
| TOC  | Return the communication time-out parameter. Value is in milliseconds   | <b>32-bit number</b> |
| TOC=[Value]                                  | Set the communication time-out parameter                                | <b>OK</b>            |
| VER  | Return the current firmware version number                              | <b>VXXX</b>          |
| V[0-63]                                      | Return the general purpose variable register.                           | <b>32-bit number</b> |
| V[0-63]=[Value]                              | Set the value to general purpose register                               | <b>OK</b>            |
| X[Value]<br>Y[Value]<br>Z[Value]<br>U[Value] | Move the indicated axis to the specified absolute/incremental position. | <b>OK</b>            |

Table 8.0

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:

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

Table 8.1

## 9. Standalone Language Specification

;

Description:

Comment notation. In programming, comment must be in its own line.

Syntax:

; [Comment Text]

Examples:

```

;***This is a comment
JOGX+           ;***Jogs X axis to positive direction
DELAY=1000      ;***Wait 1 second

```

### **ABORT**

Description:

**Motion:** Immediately stops all axes if in motion without deceleration.

Syntax:

ABORT

Examples:

```

JOGX+           ;***Jogs X axis to positive direction
DELAY=1000      ;***Wait 1 second
ABORT           ;***Stop immediately all axes including X axis

```

### **ABORT[axis]**

Description:

**Motion:** Immediately stops individual axis without deceleration.

Syntax:

ABORT[axis]

Examples:

```

JOGX+           ;***Jogs X axis to positive direction
DELAY=1000      ;***Wait 1 second
ABORTX          ;***Stop the X axis immediately

```

### **ABS**

Description:

**Motion:** Changes all move commands to absolute mode.

Syntax:

ABS

Examples:

```

ABS             ;***Change to absolute mode
PX=0            ;***Change X position to 0
X1000           ;***Move X axis to position 1000
WAITX
X2000           ;***Move X axis to position 2000
WAITX

```

## ACC

Description:

**Read:** Get acceleration value

**Write:** Set acceleration value.

Value is in milliseconds.

Range is from 1 to 10,000.

Syntax:

**Read:** [variable] = ACC

**Write:** ACC = [value]

ACC = [variable]

**Conditional:** IF ACC=[variable]  
ENDIF

IF ACC=[value]  
ENDIF

Examples:

ACC=300 ;\*\*\*Sets the acceleration to 300 milliseconds

V3=500 ;\*\*\*Sets the variable 3 to 500

ACC=V3 ;\*\*\*Sets the acceleration to variable 3 value of 500

## ACC[axis]

Description:

**Read:** Get individual acceleration value

**Write:** Set individual acceleration value.

Value is in milliseconds.

Range is from 1 to 10,000.

Syntax:

**Read:** [variable] = ACC[axis]

**Write:** ACC[axis] = [value]

ACC[axis] = [variable]

**Conditional:** IF ACC[axis]=[variable]  
ENDIF

IF ACC[axis]=[value]  
ENDIF

Examples:

ACCX=300 ;\*\*\*Sets the X acceleration to 300 milliseconds

V3=500 ;\*\*\*Sets the variable 3 to 500

ACCX=V3 ;\*\*\*Sets the X acceleration to variable 3 value of 500

## ***DELAY***

Description:

Set a delay (1 ms units)

Syntax:

Delay=[Number] (1 ms units)

Examples:

```
JOGX+           ;***Jogs X axis to positive direction
DELAY=10000     ;***Wait 10 seconds
ABORT           ;***Stop with deceleration all axes including X axis
PX=0            ;***Sets the current X position to 0
PY=0            ;***Sets the current Y position to 0
PZ=0            ;***Sets the current Z position to 0
PU=0            ;***Sets the current U position to 0
```

## ***DI***

Description:

**Read:** Gets the digital input value  
PMX-4CX-SA has 4 digital inputs

Syntax:

```
Read: [variable] = DI
Conditional: IF DI=[variable]
                ENDIF

                IF DI=[value]
                ENDIF
```

Examples:

```
IF DI=15
    DO=1           ;***If all digital inputs are triggered, set DO=1
ENDIF
```

## ***DI[1-4]***

Description:

**Read:** Gets the digital input value  
PMX-4CX-SA has 4 digital inputs

Syntax:

```
Read: [variable] = DI[1-4]
Conditional: IF DI[1-4]=[variable]
                ENDIF

                IF DI[1-4]=[0 or 1]
                ENDIF
```

Examples:

```
IF DI1=1
    DO=1           ;***If digital input 1 is triggered, set DO=1
ENDIF
```

## ***DN***

Description:

**Read:** Gets the device name

**Write:** Sets the device name

Syntax:

**Read:** [variable] = DN

**Write:** DN=[value]

DN=[variable]

**Conditional:** IF DN=[variable]  
ENDIF

IF DN=[value]  
ENDIF

Examples:

DN=2 ;\*\*\*Set device name to 4CX02

## ***DO***

Description:

**Read:** Gets the digital output value

**Write:** Sets the digital output value

PMX-4CX-SA has 4 digital outputs

Syntax:

**Read:** [variable] = DO

**Write:** DO = [value]

DO = [variable]

**Conditional:** IF DO=[variable]  
ENDIF

IF DO=[value]  
ENDIF

Examples:

DO=7 ;\*\*\*Turn first 3 bits on and rest off

## ***DO[1-4]***

Description:

**Read:** Gets the individual digital output value

**Write:** Sets the individual digital output value

PMX-4CX-SA has 4 digital outputs

Syntax:

**Read:** [variable] = DO[1-4]

**Write:** DO[1-4] = [0 or 1]

DO[1-4] = [variable]

**Conditional:** IF DO[1-4]=[variable]  
ENDIF

IF DO[1-4]=[0 or 1]

ENDIF

Examples:

```
DO2=1      ;***Turn DO2 on
DO4=1      ;***Turn DO4 on
```

### **ELSE**

Description:

Perform ELSE condition check as a part of IF statement

Syntax:

ELSE

Examples:

```
IF V1=1
    X1000      ;***If V1 is 1, then move to 1000
    WAITX
ELSE
    X-1000     ;***If V1 is not 1, then move to -1000
    WAITX
ENDIF
```

### **ELSEIF**

Description:

Perform ELSEIF condition check as a part of the IF statement

Syntax:

ELSEIF [Argument 1] [Comparison] [Argument 2]

[Argument] can be any of the following:

- Numerical value
- Digital Output
- Digital Input
- Enable Output
- Motor Status

[Comparison] can be any of the following

- = Equal to
- > Greater than
- < Less than
- >= Greater than or equal to
- <= Less than or equal to
- != Not Equal to

Examples:

```
IF V1=1
    X1000
    WAITX
ELSEIF V1=2
    X2000
    WAITX
ENDIF
```

## **END**

### Description:

Indicate end of program.

Program status changes to idle when END is reached.

**Note:** Subroutine definitions should be written AFTER the END statement

### Syntax:

END

### Examples:

```
X0
WAITX
X1000
WAITX
END
```

## **ENDIF**

### Description:

Indicates end of IF operation

### Syntax:

ENDIF

### Examples:

```
IF V1=1
    X1000
    WAITX
ENDIF
```

## **ENDSUB**

### Description:

Indicates end of subroutine

When ENDSUB is reached, the program returns to the previous subroutine.

### Syntax:

ENDSUB

### Examples:

```
GOSUB 1
END

SUB 1
    X0
    WAITX
ENDSUB
```

## **ENDWHILE**

### Description:

Indicate end of WHILE loop

### Syntax:

ENDWHILE

### Examples:

```

WHILE V1=1      ;***While V1 is 1 continue to loop
  X0
  WAITX
  X1000
  WAITX
ENDWHILE      ;***End of while loop so go back to WHILE

```

## **EO**

Description:

**Read:** Gets the enable output value

**Write:** Sets the enable output value

Performax 4CX has 4 enable outputs.

Syntax:

**Read:** [variable] = EO

**Write:** EO = [value]

EO = [variable]

**Conditional:** IF EO=[variable]

ENDIF

IF EO=[value]

ENDIF

Examples:

```

EO=3      ;***Turn first 2 bits of enable outputs
IF V1=1
  EO=V2    ;***Enable output according to variable 2
ENDIF

```

## **EO[1-4]**

Description:

**Read:** Gets the individual enable output value

**Write:** Sets the individual enable output value

Performax 4CX has 4 enable outputs.

Syntax:

**Read:** [variable] = EO[1-4]

**Write:** EO[1-4] = [0 or 1]

EO[1-4] = [variable]

**Conditional:** IF EO=[variable]

ENDIF

IF EO=[value]

ENDIF

Examples:

```

EO1=31     ;***Turn enable output 1 on
IF V1=1
  EO2=V2    ;***Enable output 2 according to variable 2
ENDIF

```

## **GOSUB**

### Description:

Perform go to subroutine operation

Subroutine range is from 1 to 31.

**Note:** Subroutine definitions should be written AFTER the END statement  
SUB 31 is reserved for error handling.

### Syntax:

GOSUB [subroutine number]

[Subroutine Number] range is 1 to 31

### Examples:

```
GOSUB 1
```

```
END
```

```
SUB 1
```

```
    X0
```

```
    WAITX
```

```
ENDSUB
```

## **HLHOME[axis][+ or -]**

### Description:

**Command:** Perform low speed homing using current high speed, low speed, and acceleration.

### Syntax:

HLHOME[Axis][+ or -]

### Examples:

```
HLHOMEX+ ;***Low speed homes X axis in positive direction
```

```
WAITX
```

```
HLHOMEZ- ;***Low speed homes Z axis in negative direction
```

```
WAITZ
```

## **HOME[axis][+ or -]**

### Description:

**Command:** Perform homing using current high speed, low speed, and acceleration.

### Syntax:

HOME[Axis][+ or -]

### Examples:

```
HOMEX+ ;***Homes X axis in positive direction
```

```
WAITX
```

```
HOMEZ- ;***Homes Z axis in negative direction
```

```
WAITZ
```

## **HSPD**

### Description:

**Read:** Gets high speed. Value is in pulses/second  
**Write:** Sets high speed. Value is in pulses/second.  
 Range is from 1 to 300,000.

Syntax:

**Read:** [variable] = HSPD  
**Write:** HSPD = [value]  
           HSPD = [variable]  
**Conditional:** IF HSPD=[variable]  
                   ENDIF  
  
                   IF HSPD=[value]  
                   ENDIF

Examples:

HSPD=10000 ;\*\*\*Sets the high speed to 10,000 pulses/sec  
 V1=2500 ;\*\*\*Sets the variable 1 to 2,500  
 HSPD=V1 ;\*\*\*Sets the high speed to variable 1 value of 2500

### ***HSPD[axis]***

Description:

**Read:** Gets individual high speed. Value is in pulses/second  
**Write:** Sets individual high speed. Value is in pulses/second.  
 Range is from 1 to 300,000.

Syntax:

**Read:** [variable] = HSPD[axis]  
**Write:** HSPD[axis] = [value]  
           HSPD[axis] = [variable]  
**Conditional:** IF HSPD[axis]=[variable]  
                   ENDIF  
  
                   IF HSPD[axis]=[value]  
                   ENDIF

Examples:

HSPDY=10000 ;\*\*\*Sets the Y high speed to 10,000 pulses/sec  
 V1=2500 ;\*\*\*Sets the variable 1 to 2,500  
 HSPDY=V1 ;\*\*\*Sets the Y high speed to variable 1 value of 2500

### ***IF***

Description:

Perform IF condition check

Syntax:

IF [Argument 1] [Comparison] [Argument 2]

[Argument] can be any of the following:

Numerical value  
 Pulse or Encoder Position  
 Digital Output

Digital Input  
 Enable Output  
 Motor Status

[Comparison] can be any of the following

= Equal to  
 > Greater than  
 < Less than  
 >= Greater than or equal to  
 <= Less than or equal to  
 != Not Equal to

Examples:

```
IF V1=1
  X1000
  WAITX
ENDIF
```

### ***INC***

Description:

**Command:** Changes all move commands to incremental mode.

Syntax:

INC

Examples:

```
INC ;***Change to incremental mode
PX=0 ;***Change X position to 0
X1000 ;***Move X axis to position 1000 (0+1000)
WAITX
X2000 ;***Move X axis to position 3000 (1000+2000)
WAITX
ABORT ;***Stop immediately all axes including X axis
```

### ***JOG[axis]***

Description:

**Command:** Perform jogging using current high speed, low speed, and acceleration.

Syntax:

JOG[Axis][+ or -]

Examples:

```
JOGX+ ;***Jogs X axis in positive direction
JOGY- ;***Jogs Y axis in negative direction
```

### ***LHOME[axis][+ or -]***

Description:

**Command:** Perform limit homing using current high speed, low speed, and acceleration.

Syntax:

LHOME[Axis][+ or -]

Examples:

```
LHOMEX+ ;***Limit homes X axis in positive direction
WAITX
LHOMEZ- ;***Limit homes Z axis in negative direction
WAITZ
```

## **LSPD**

Description:

**Read:** Get low speed. Value is in pulses/second.

**Write:** Set low speed. Value is in pulses/second.

Range is from 1 to 300,000.

Syntax:

**Read:** [variable]=LSPD

**Write:** LSPD=[long value]

LSPD=[variable]

**Conditional:** IF LSPD=[variable]

ENDIF

IF LSPD=[value]

ENDIF

Examples:

```
LSPD=1000 ;***Sets the start low speed to 1,000 pulses/sec
V1=500 ;***Sets the variable 1 to 500
LSPD=V1 ;***Sets the start low speed to variable 1 value of 500
```

## **LSPD[axis]**

Description:

**Read:** Get individual low speed. Value is in pulses/second.

**Write:** Set individual low speed. Value is in pulses/second.

Range is from 1 to 300,000.

Syntax:

**Read:** [variable]=LSPD[axis]

**Write:** LSPD[axis]=[long value]

LSPD[axis]=[variable]

**Conditional:** IF LSPD[axis]=[variable]

ENDIF

IF LSPD[axis]=[value]

ENDIF

Examples:

```
LSPDZ=1000 ;***Sets the Z low speed to 1,000 pulses/sec
V1=500 ;***Sets the variable 1 to 500
LSPDZ=V1 ;***Sets the Z low speed to variable 1 value of 50
```

### ***MIO[axis]***

Description:

**Command:** Get motor input/output status of axis

Syntax:

MIO[Axis]

Examples:

```
IF MIOX=0
    DO=6
ELSEIF MIOY=0
    DO=3
ELSEIF MIOZ=0
    DO=2
ELSEIF MIOU=0
    DO=1
ENDIF
```

### ***MST[axis]***

Description:

**Command:** Get motor status of axis

Syntax:

MST[Axis]

Examples:

```
IF MSTX=0
    DO=6
ELSEIF MSTY=0
    DO=3
ELSEIF MSTZ=0
    DO=2
ELSEIF MSTU=0
    DO=1
ENDIF
```

### ***P[axis]***

Description:

**Read:** Gets the current pulse position

**Write:** Sets the current pulse position

Syntax:

**Read:** Variable = P[axis]

**Write:** P[axis] = [value]

P[axis] = [variable]

**Conditional:** IF P[axis]=[variable]

ENDIF

IF P[axis]=[value]

ENDIF

Examples:

```
JOGX+           ;***Jogs X axis to positive direction
DELAY=1000      ;***Wait 1 second
ABORT           ;***Stop with deceleration all axes including X axis
PX=0            ;***Sets the current pulse position to 0
```

### ***PS[axis]***

Description:

**Read:** Get the current pulse speed of an axis

Syntax:

```
Read: Variable = PS[Axis]
Conditional: IF PS[axis]=[variable]
                ENDIF
                IF PS[axis]=[value]
                ENDIF
```

Examples:

```
JOGX+           ;***Jogs X axis to positive direction
DELAY=1000      ;***Wait 1 second
ABORT           ;***Stop with deceleration all axes including X axis
V1=PSX          ;***Sets variable 1 to pulse speed X
JOGY+           ;***Jogs Y axis to positive direction
V2=PSY          ;***Sets variable 2 to pulse speed Y
```

### ***SR[0,3]***

Description:

**Write:** Set the standalone control for the specified standalone program

Syntax:

```
Write: SR[0-3] = [0-3]
        SR[0-3] = [0-3]
```

Examples:

```
IF DI1=1        ; If digital input 1 is on
    SR0=0       ; Turn off standalone program 0
ENDIF
```

### ***STOP***

Description:

**Command:** Stop all axes if in motion with deceleration.  
Previous acceleration value is used for deceleration.

Syntax:

```
STOP
```

Examples:

```
JOGX+           ;***Jogs X axis to positive direction
DELAY=1000      ;***Wait 1 second
STOP            ;***Stop with deceleration all axes including X axis
```

## **STOP[axis]**

### Description:

Stop individual axis if in motion with deceleration.  
Previous acceleration value is used for deceleration.

### Syntax:

STOP[axis]

### Examples:

```
JOGX+           ;***Jogs X axis to positive direction
DELAY=1000      ;***Wait 1 second
JOGY+           ;***Jogs Y axis to positive direction
DELAY=1000      ;***Wait 1 second
STOPX           ;***Stop with deceleration X axis only
```

## **STORE**

### Description:

Store device settings and second half of general purpose variable registers (V32-V63) to flash.

### Syntax:

STORE

### Example:

```
V35=100
V36=200
STORE           ;***Values of V35 and V36 will now be preserved after power
cycle
```

## **SUB**

### Description:

Indicates start of subroutine

### Syntax:

SUB [subroutine number]  
[Subroutine Number] range is 0 to 31  
**Note:** SUB 31 is reserved for error handling.

### Examples:

```
GOSUB 1
END
SUB 1
    X0
    WAITX
    X1000
    WAITX
ENDSUB
```

## **TOC**

### Description:

Sets the communication time-out parameter. Value is in milli-seconds.

Syntax:

TOC=[long value]

Examples:

TOC=10000 ;\*\*\*Sets time-out parameter to 10 seconds

## U

Description:

**Command:** Perform U axis move to target location

Syntax:

U[value]

U[variable]

Examples:

U10000 ;\*\*\*Move U Axis to position 10000

V10 = 1200 ;\*\*\*Set variable 10 value to 1200

UV10 ;\*\*\*Move U Axis to variable 10 value

## V[index]

Description:

Assign to variable.

Performax 4CX has 64 variables [V0-V63]

Syntax:

V[Variable Number] = [Argument]

V[Variable Number] = [Argument1][Operation][Argument2]

*Special case for BIT NOT:*

V[Variable Number] = ~[Argument]

[Argument] can be any of the following:

- Numerical value
- Pulse or Encoder Position
- Digital Output
- Digital Input
- Enable Output
- Motor Status

[Operation] can be any of the following

- + Addition
- Subtraction
- \* Multiplication
- / Division
- % Modulus
- >> Bit Shift Right
- << Bit Shift Left
- & Bit AND
- | Bit OR
- ~ Bit NOT

Examples:

V1=12345 ;\*\*\*Set Variable 1 to 123

```
V2=V1+1      ;***Set Variable 2 to V1 plus 1
V4=DO        ;***Sets Variable 4 to digital output value
V5=~EO       ;***Sets Variable 5 to bit NOT of enable output value
```

*Note: On the STORE command, the second half of general purpose variable registers (V32-V63) are stored to flash. Their values will be preserved after power cycle.*

### **WAIT[axis]**

Description:

Tell program to wait until move on the certain axis is finished before executing next line.

Syntax:

```
WAIT[axis]
```

Examples:

```
X10000      ;***Move X Axis to position 10000
WAITX       ;***Wait until X Axis move is done
DO=5        ;***Set digital output
Y3000       ;***Move Y Axis to 3000
WAITY       ;***Wait until Y Axis move is done
```

### **WHILE**

Description:

Perform WHILE loop

Syntax:

```
WHILE [Argument 1] [Comparison] [Argument 2]
```

[Argument] can be any of the following:

- Numerical value
- Pulse or Encoder Position
- Digital Output
- Digital Input
- Enable Output
- Motor Status

[Comparison] can be any of the following

- = Equal to
- > Greater than
- < Less than
- >= Greater than or equal to
- <= Less than or equal to
- != Not Equal to

Examples:

```
WHILE V1=1      ;***While V1 is 1 continue to loop
  X0
  X1000
ENDWHILE
```

## X

Description:

**Command:** Perform X axis move to target location

Syntax:

X[value]  
X[variable]

Examples:

```
X10000      ;***Move X Axis to position 10000
WAITX
Y3000      ;***Move Y to 3000
WAITY
V10 = 1200 ;***Set variable 10 value to 1200
XV10      ;***Move X Axis to variable 10 value
WAITX
```

## Y

Description:

**Command:** Perform Y axis move to target location

Syntax:

Y[value]  
Y[variable]

Examples:

```
Y10000      ;***Move Y Axis to position 10000
WAITY
Z3000      ;***Move Z to 3000
WAITZ
V10 = 1200 ;***Set variable 10 value to 1200
YV10      ;***Move Y Axis to variable 10 value
WAITY
```

## Z

Description:

**Command:** Perform Z axis move to target location

Syntax:

Z[value]  
Z[variable]

Examples:

```
Z10000      ;***Move Z Axis to position 10000
WAITX
Y1000      ;***Move Y to 1000
WAITY
V10 = 1200 ;***Set variable 10 value to 1200
ZV10      ;***Move Z Axis to variable 10 value
WAITZ
```

## 10. Example Standalone Programs

### ***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
```

### ***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
```

### ***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
```

### **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

```

### **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    ;* 1000 increment is not continuously done
ENDSUB

```

### **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 move to complete
  ENDIF
  V1=MSTX      ;* Store the motor status to variable 1
  V2=V1&7      ;* Get first 3 bits
  IF V2!=0
    DO1=1
  ELSE
    DO1=0
  ENDIF
ENDWHILE       ;* Go back to WHILE statement
END

```

### **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 20000pps
LSPD=500                              ;* Set low speed to 500pps
ACC=500                               ;* Set acceleration to 500ms
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                             ;* End if statements
ENDWHILE                              ;* Go back to WHILE statement
END                                    ;* End Program 1

```

### **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 pps
LSPD=500                              ;* Set low speed to 500pps
ACC=500                               ;* Set acceleration to 500ms
TOC=5000                              ;* Set time-out counter 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 = 1024                     ;* 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

```

## **Contact Information**

Arcus Technology, Inc.

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

[www.arcustechnology.com](http://www.arcustechnology.com)

The information in this document is believed to be accurate at the time of publication but is subject to change without notice.