DMX-UMD

Integrated Step Motor Encoder/Driver/Controller with USB 2.0/RS-485 communication

First edition, January 2008

Revision History:

1.10 – 1st Release
1.15 – 2nd Release
1.16 – 3rd Release
1.18 – 4th Release
1.19 – 5th Release

Firmware Compatibility:

[1]V231BL

[1]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

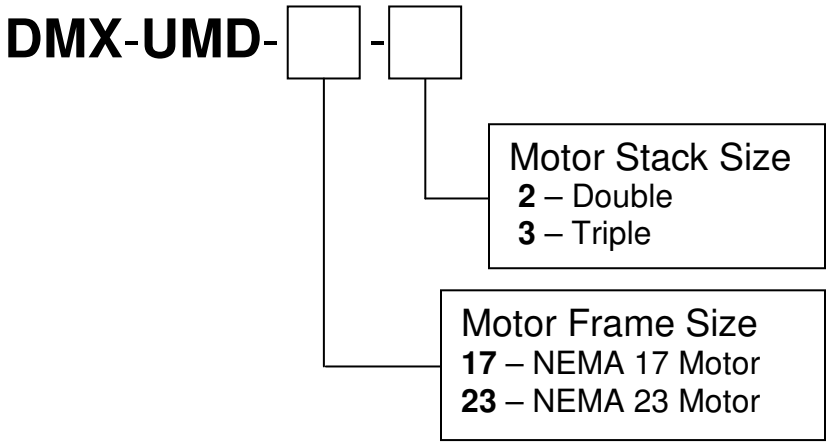
DMX-UMD is an integrated stepper controller + driver + motor motion product. Communication to the DMX-UMD can be established over USB or RS-485. It is also possible to download a stand-alone program to the device and have it run independent of a host.

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

1.1. Features

- USB 2.0 communication
- RS-485 ASCII communication
  - 9600, 19200, 38400, 57600, 115200 bps
- Standalone programmable using A-SCRIPT
- Digital IO communication
  - 4 bit motion profile select inputs (DI3-DI6)
  - One start motion input (DI1)
  - One abort/clear motion input (DI2)
  - One in position output (DO1)
  - One error output (DO2)
- A/B/Z differential encoder
  - StepNLoop closed loop control (position verification)
  - 1000 line incremental encoder (4000 counts/rev with 4x quadrature decoding)
- Opto-isolated I/O
  - 6 x inputs
  - 2 x outputs
  - 1 x High speed position capture latch input
  - +Limit/-Limit/Home inputs
- Homing routines:
  - Home input only
  - Limit only
  - Z-index encoder channel only
  - Home input + Z index encoder channel
- S-curve or trapezoidal acceleration profile control
- On-the-fly speed change
- Stepper driver
  - 12-48 VDC
  - 3.0 Amp max current setting (peak current)
  - 2 to 500 micro-step setting
  - 1 MHz max pulse support
- Stepper motor
  - NEMA 17/23 motor sizes available in different stack sizes

1.2. Part Numbering Scheme



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

2. Electrical and Thermal Specifications

Parameter	Min	Max	Units
Main Power Input	+12	+48	V
	-	3.0	A
Digital Inputs (DI, Home, Lim, Latch)	+12	+24	V
	-	45	mA
Digital Outputs (DO)	-	+24	V
	-	90[1]	mA
Operating Temperature [2]	-20	80	℃
Storage Temperature [2]	-55	150	℃

Table 2.0

[1] A current limiting resistor is required.
[2] Based on component ratings.

3. Dimensions

3.1. DMX-UMD-17

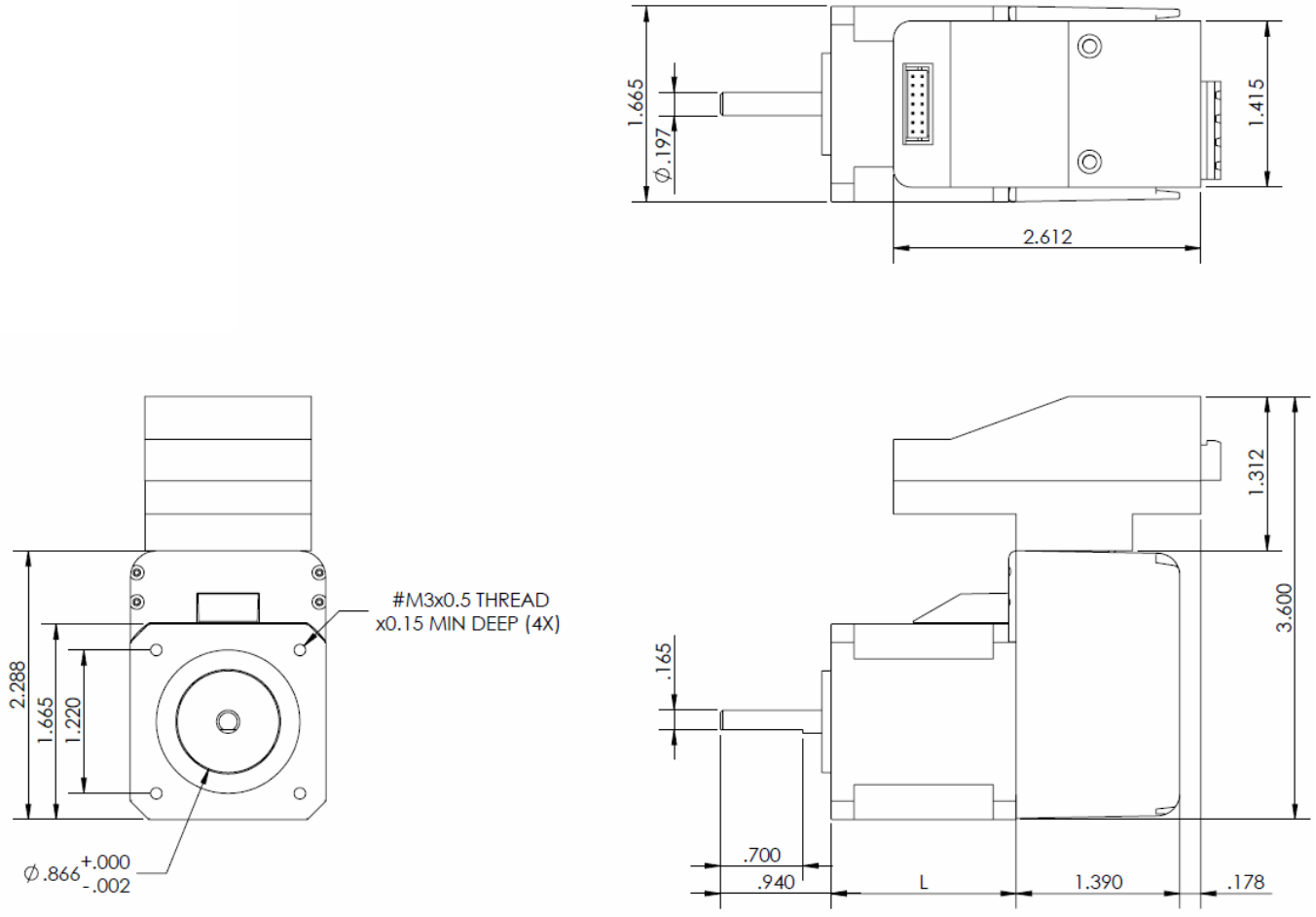


Figure 3.0

NEMA 17 Models	L (inches)
DMX-UMD-17-2 (double stack)	1.58
DMX-UMD-17-3 (triple stack)	1.89

Table 3.0

3.2. DMX-UMD-23

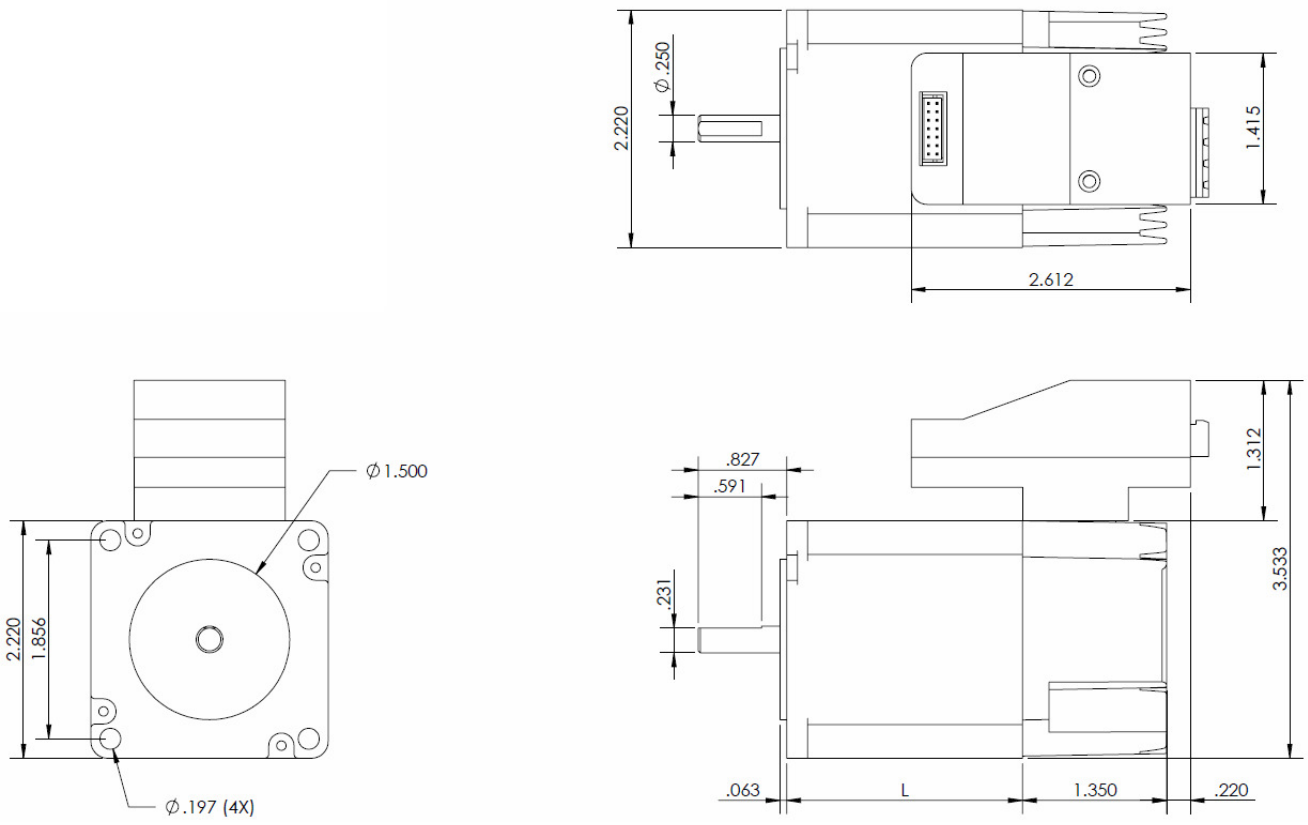


Figure 3.1

NEMA 23 Models	L (inches)
DMX-UMD-23-2 (double stack)	2.20
DMX-UMD-23-3 (triple stack)	2.99

Table 3.1

4. Connectivity

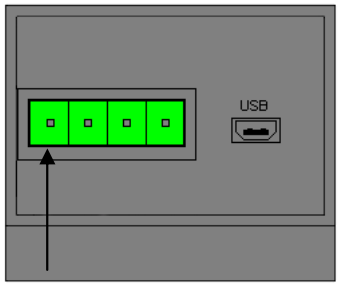


Figure 4.0

4.1. 4-Pin Connector (5.08mm)

Pin #	In/Out	Name	Description
1	I/O	485-	RS-485 minus signal
2	I/O	485+	RS-485 plus signal
3	I	GND	Ground
4	I	V+	Power Input +12 to +48VDC

Table 4.0

Mating Connector Description: 4 pin 0.2” (5.08mm) connector
Mating Connector Manufacturer: On-Shore
Mating Connector Manufacturer Part: [1]EDZ950/4

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

4.2. 14-Pin Connector (2mm)

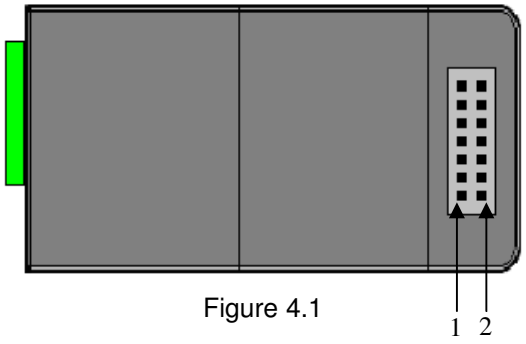


Figure 4.1

Pin #	Wire Color	In/Out	Name	Description
1	Orange	I	OPTO	+12 to +24VDC opto-supply input – used for limit, home, latch, and digital inputs
2	Orange	I	OPTO	+12 to +24VDC opto-supply input – used for limit, home, latch, and digital inputs

3	Yellow	I	+LIM	Plus limit input
4	Yellow/White	I	-LIM	Minus limit input
5	White	I	HOME	Home input
6	Orange/Yellow	I	LATCH	Latch input
7	Brown/Yellow	I	DI1	Digital Input 1
8	Brown/Yellow	I	DI2	Digital Input 2
9	Brown/Yellow	I	DI3	Digital Input 3
10	Brown/Yellow	I	DI4	Digital Input 4
11	Brown/Yellow	I	DI5	Digital Input 5
12	Brown/Yellow	I	DI6	Digital Input 6
13	Black/Yellow	O	DO1	Digital Output 1
14	Black/Yellow	O	DO2	Digital Output 2

Table 4.1

Mating Connector Description: 14 pin 2mm dual row connector
Mating Connector Manufacturer: HIROSE
Mating Connector Housing Part Number: DF11-14DS-2C
Mating Connector Pin Part Number: DF11-2428SC

4.3. DMX-UMD Interface Circuit

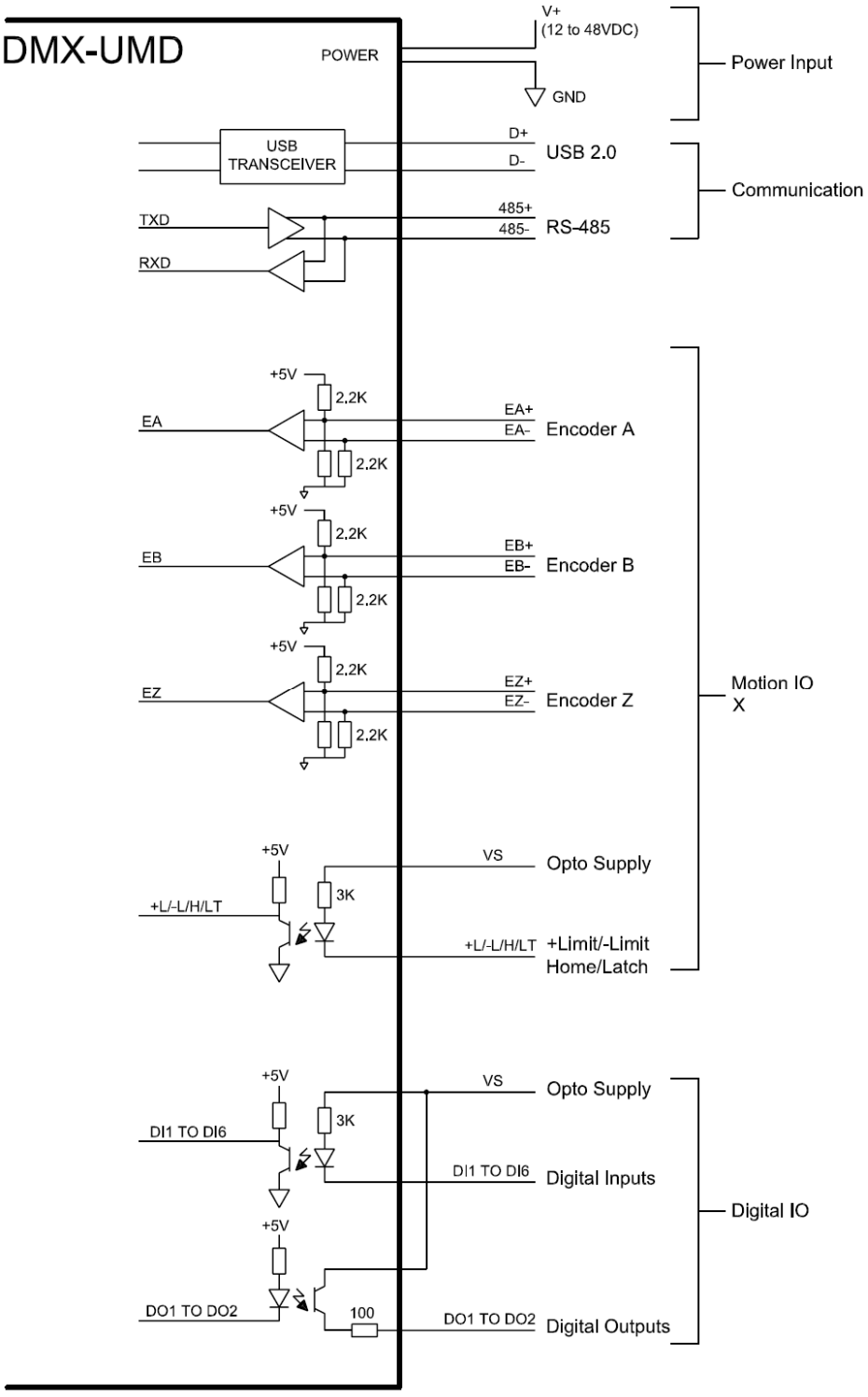


Figure 4.2

4.4. Digital Inputs

Figure 4.3 shows the detailed schematic of the opto-isolated limit, home, and general purpose inputs. All opto-isolated digital inputs are NPN type.

.

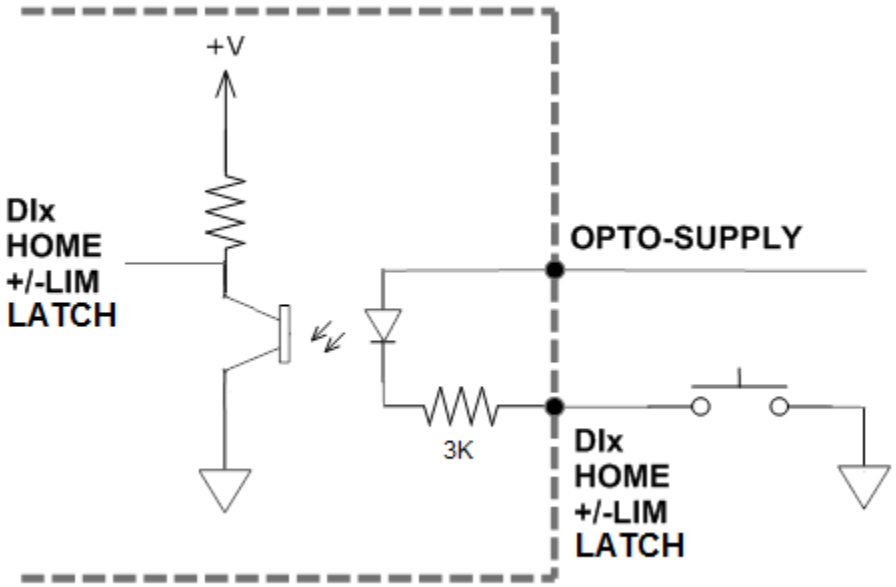


Figure 4.3

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 digital 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.

4.5. Digital Outputs

Figure 4.4 shows an example wiring to the digital output. All opto-isolated digital outputs will be PNP type.

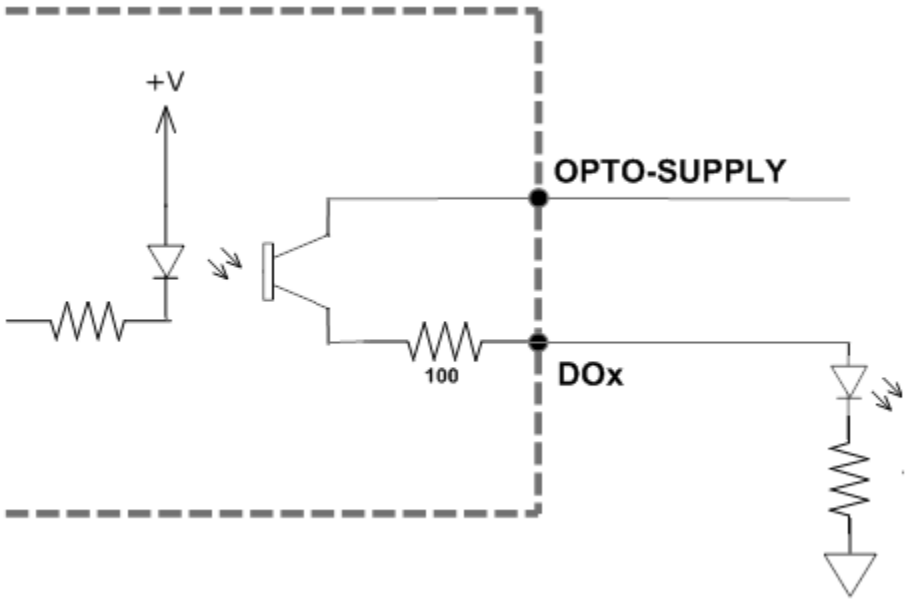


Figure 4.4

The opto-supply must be connected to +12 to +24VDC in order for the digital outputs to operate.

When activated, the opto-isolator for the digital output pulls the voltage on the digital output line to the opto-supply. The maximum source 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 opto-supply.

5. Stepper Motor Driver Overview

5.1. Microstep

The standard DMX-UMD motor is a 1.8 degree motor, which translates to 200 full steps per revolution. These steps can be divided with microstepping to increase position resolution. DMX-UMD comes with a bipolar step motor and has configurable microstep setting range from 2 to 500 microsteps.

5.2. Driver Current

The DMX-UMD will have a maximum rated driver current that is dependent on the stack size of the motor. See table 5.0 for details.

Setting the driver current higher than the maximum rated current will overheat the motor and driver and potentially damage the unit. It is recommended to use a current setting that is in the range of 60-80% of the maximum rated current for the motor.

DMX-UMD has configurable current setting from 100mA to 3.0A. Driver current is set to the "Run Current" setting whenever the motor is moving. Similarly, the driver current is set to the "Idle Current" setting when the motor is idle for a period of time longer than the "Idle Time" setting. See section 7.18 for more details regarding the available driver settings.

The Run Current and the Idle Current should not go over the maximum rated current for each motor size. Use table 5.0 as a reference on maximum rated current setting.

Product	Maximum Peak Rated Driver Current Setting (Amp)
DMX-UMD	1.7
DMX-UMD	2.0
DMX-UMD	3.0
DMX-UMD	3.0

Table 5.0

5.3. Operating Temperature

Electronic components used in the DMX-UMD have a maximum ambient operating temperature of **85 C**˚. DMX-UMD electronics are potted with heat-conductive compound to the housing to evenly distribute the heat and reduce any hot spots in the driver. The housing also has integrated fins to better dissipate the heat.

DMX-UMD should be mounted securely to a metallic bracket that can also act as a heat-sink. During operation, the step motor section typically becomes hotter than the driver section. Having the step motor mounted to a heat sink will help dissipate the heat generated by the step motor.

DMX-UMD mounting orientation should be such that the fins are oriented vertically for better convection and heat dissipation. See figure 5.0 below.

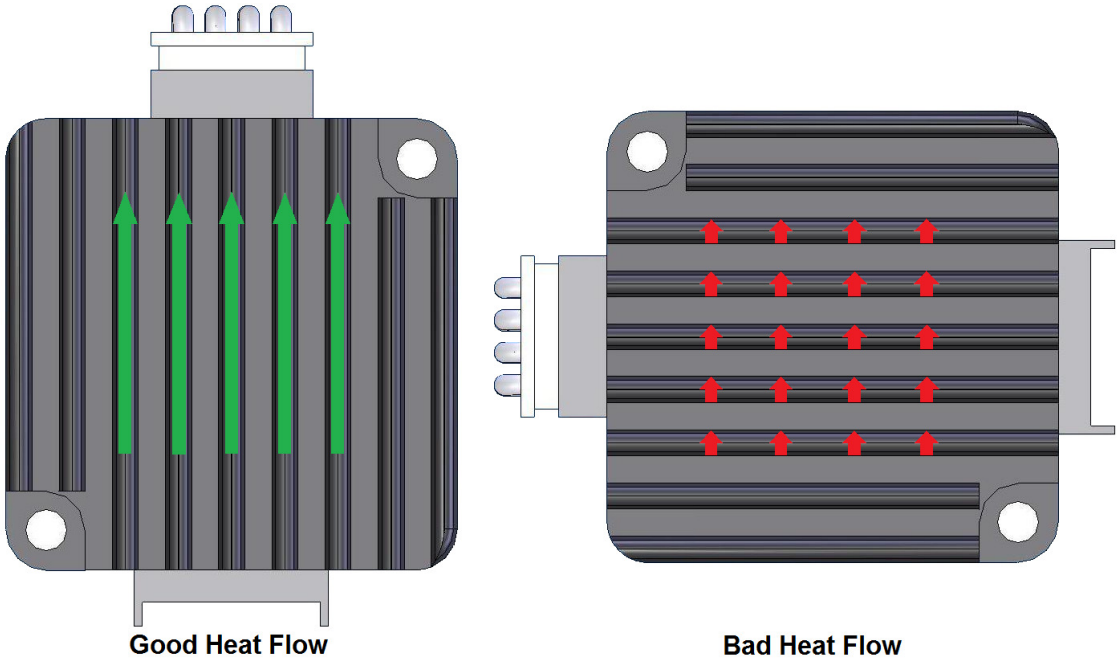


Figure 5.0

DMX-UMD has a temperature sensor to detect over heating of the driver. Temperature sensing is done only when the driver is enabled. When the temperature goes over the over-temperature alarm value 70 C˚, the Alarm Output is turned on. If the temperature goes below the 68 C˚, the alarm output is turned off. If the temperature goes over 75 C˚, the driver is automatically turned off and remained off until the temperature goes below 68 C˚.

For details on the over temperature alarm, see figure 5.1.

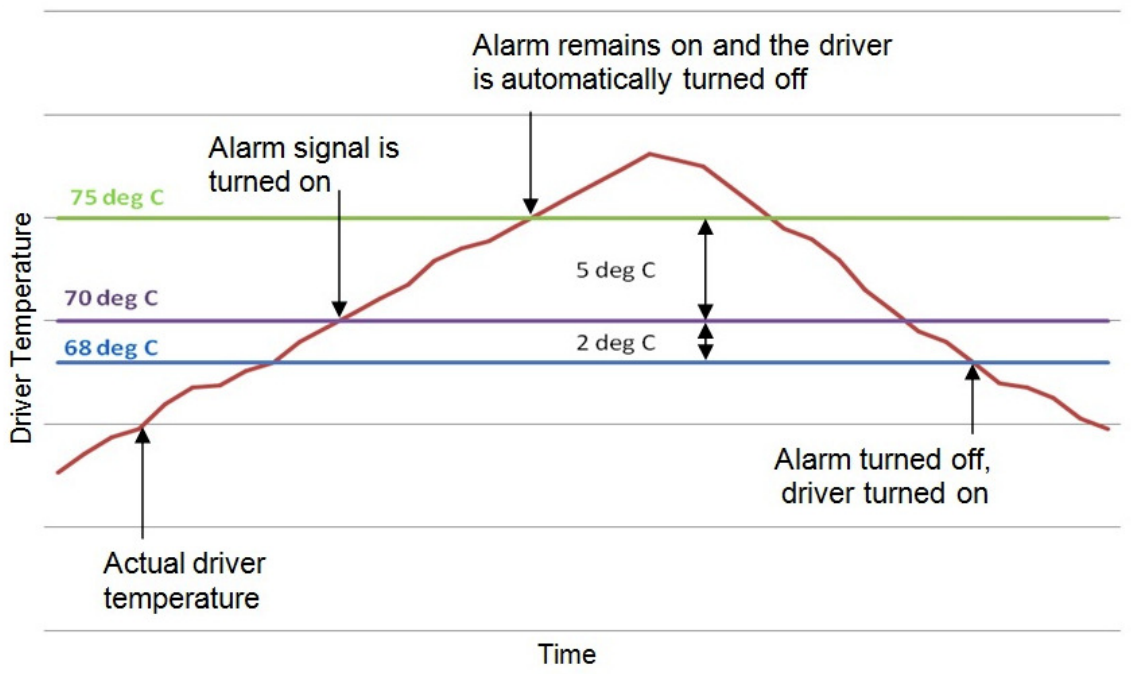


Figure 5.1

5.4. Stepper Motor Specifications

Following chart shows the specifications of standard step motors used for DMX-UMD products. All standard DMX-UMD step motors are 1.8 degree bi-polar step motors.

NEMA Size	Stack Size	Max Amp / Phase	Resistance / Phase	Inductance / Phase	Inertia
17	Double	1.7A	1.5 Ω	3.0 mH	0.28 oz-in^2
	Triple	2.0A	1.4 Ω	2.7 mH	0.37 oz-in^2
23	Double	2.8A	0.9 Ω	2.5 mH	1.64 oz-in^2
	Triple	2.8A	1.13 Ω	3.6 mH	2.62 oz-in^2

Table 5.1

NEMA Size	Stack Size	Max Axial Force	Max Radial Force
17	Double	15N	10N
	Triple	15N	10N
23	Double	15N	75N
	Triple	15N	75N

Table 5.2

5.5. Stepper Motor Torque

The torque output of the DMX-UMD will vary depending on the supply voltage, driver current, motor type, and target speed of the motor.

Increasing the drive current will increase the torque output, however the operating temperature will also increase. While decreasing the drive current will reduce the torque output, it will help reduce the operating temperature as well. Each application will need to adjust this setting to find the desired driver output.

Using a higher voltage to power the DMX-UMD will allow the motor to run at faster speeds. Note that increasing the voltage will not increase the maximum holding torque of the motor.

Stepper motors in general will drop off in torque as the target speed of the motor is increased. The following torque curve shows the expected torque output based on the motor speed of the DMX-UMD.

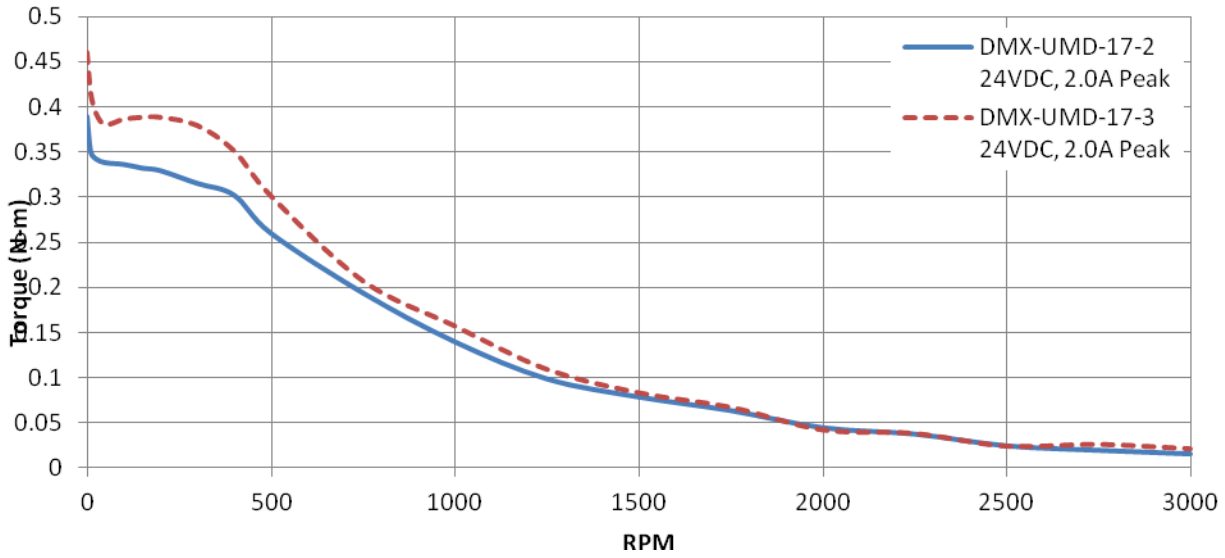


Figure 5.2

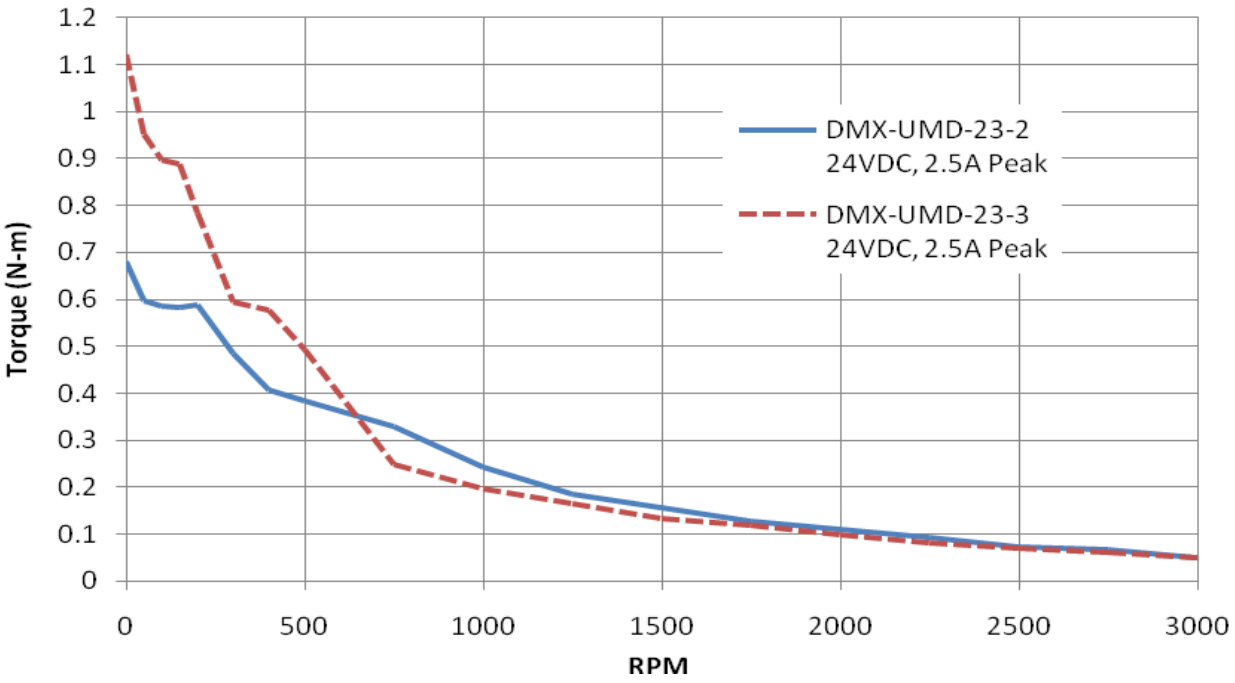

1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Torque (N-m)
0
500
1000
1500
2000
2500
3000
RPM
DMX-UMD-23-2
24VDC, 2.5A Peak
DMX-UMD-23-3
24VDC, 2.5A Peak

Figure 5.3

6. Communication Interface

6.1. USB Communication (ASCII)

DMX-UMD USB communications is USB 2.0 compliant.

In order to communicate with DMX-UMD via USB, the proper driver must be first installed. Before connecting the DMX-UMD device or running any program, please go to the Arcus web site, download the USB driver installation instructions and run the Arcus Drivers and Tools Setup.

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

6.1.1. Typical USB Setup

The DMX-UMD 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 6.0.

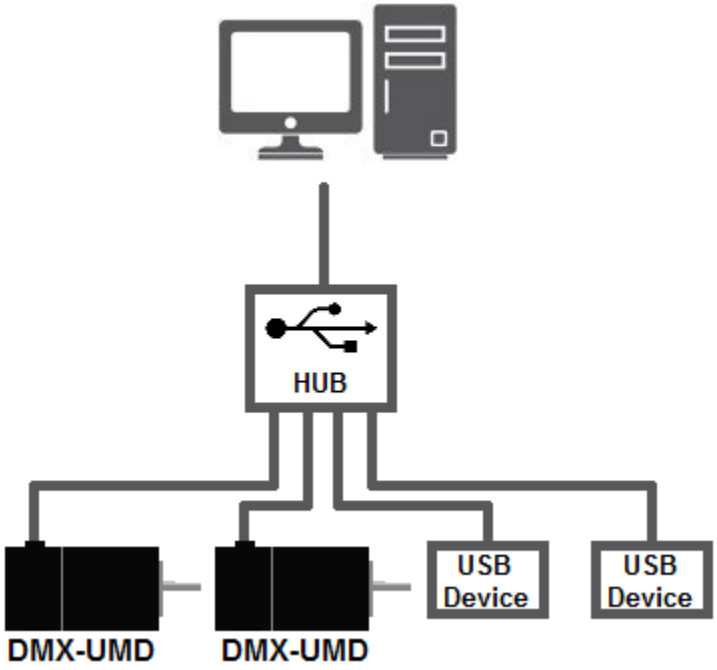


Figure 6.0

6.1.2. API Functions

Communication between the PC and DMX-UMD is done using Windows compatible DLL API function calls as shown below. Windows programming language 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 DMX-UMD for sending a command from a PC and getting a reply from DMX-UMD 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.

6.1.3. 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 photo below:

Figure 6.1

6.2. RS-485 Communication (ASCII)

6.2.1. Typical RS-485 Setup

A typical RS-485 network is shown in figure 6.2. Several techniques can be used to increase the robustness of an RS-485 network. Please see section 6.2.5 for details.

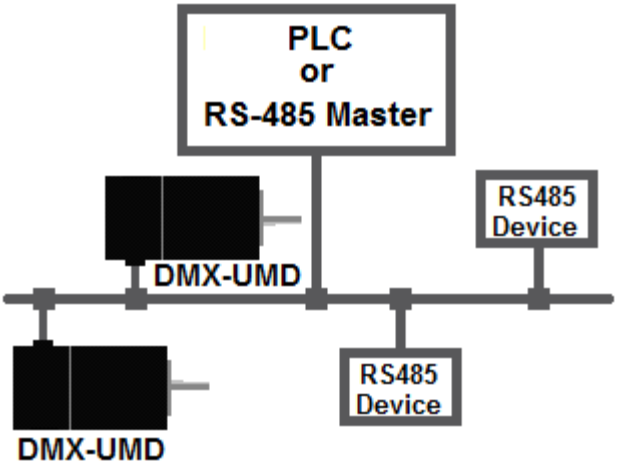


Figure 6.2

6.2.2. Communication Port Settings

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

Table 6.0

DMX-UMD 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 6.1

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 before a power cycle is completed, the settings will not take effect.

By default, the DMX-UMD has a baud rate setting of 9600 bps.

6.2.3. ASCII Protocol

Sending Command

ASCII command string in the format of

@[DeviceName][ASCII Command][CR]

Receiving Reply

The response will be in the format of

[Response][CR]

[CR] character has ASCII code 13.

Examples:

For querying the x-axis polarity

Send: @00POL[CR]
Reply (if RT=0): 7[CR]
Reply (if RT=1): #007[CR]

For jogging the x-motor in positive direction

Send: @00J+[CR]
Reply (if RT=0): OK[CR]
Reply (if RT=1): #00OK[CR]

For aborting any motion in progress

Send: @00ABORT[CR]
Reply (if RT=0): OK[CR]
Reply (if RT=1): #00OK[CR]

RT is a parameter that sets the response type of the device. See section 6.2.4 for details.

6.2.4 Response Type

It is possible to choose between two types of response string formats. This parameter can be set using the **RT** command.

Format 1 (default): [Response][CR]

Examples:
For querying the encoder position
Send: @01EX[CR]
Reply: 1000[CR]

For jogging the motor in positive direction
Send: @01J+[CR]
Reply: OK[CR]

To achieve this response string type, set **RT=0**.

Format 2: #[DeviceName][Response][CR]
Examples:
For querying the encoder position
Send: @01EX[CR]
Reply: #011000[CR]

For jogging the motor in positive direction
Send: @01J+[CR]
Reply: #01OK[CR]

To achieve this response string type, set **RT=1**.

To write the response type parameter to flash memory, use the **STORE** command. After a complete power cycle, the new response type will take effect. Note that before a power cycle is done, the setting will not take effect.

6.2.5. Broadcasting over RS-485

The address '00' is reserved for broadcasting over an RS-485 bus. Any ASCII command prefixed by '@00' will be processed by all DMX-UMD modules on the RS-485 bus. When a broadcast command is received by an DMX-UMD module, no response is sent back to the master.

6.2.6. 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.

1. Daisy Chaining: For a multi-drop RS-485 network, be sure that the network uses daisy-chain wiring. Figure 6.2 shows an example of a daisy chain network.
2. Number of Nodes: The maximum number of nodes recommended is 32. Increasing beyond this number will require special attention
3. Twisted Pair Wiring: 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.
4. Termination: 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.

6.3. Device Number

If multiple DMX-UMD devices are connected to the PC, each device should have a unique device number. This will allow the PC to differentiate between multiple motors. In order to make this change to a DMX-UMD, first store the desired number using the **DN** command. Note that this value must be within the range [UMD00, UMD99].

For example, to change the device number, the command **DN=UMD02** can be sent. The device name can also be changed through the Setup window of the standard DMX-UMD software. See section 8 for details.

By default, all DMX-UMD start with device number UMD01.

To save a modified device number to the DMX-UMD's flash memory, 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.

6.4. Communication Time-out Feature (Watchdog)

DMX-UMD 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 10 of the **MST** parameter is turned on. The time-out value is set by the **TOC** command. Units are in milliseconds. This feature is usually used in standalone mode. Refer to the section 10 for an example.

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

6.5. DIO Communication

6.5.1. Typical DIO setup

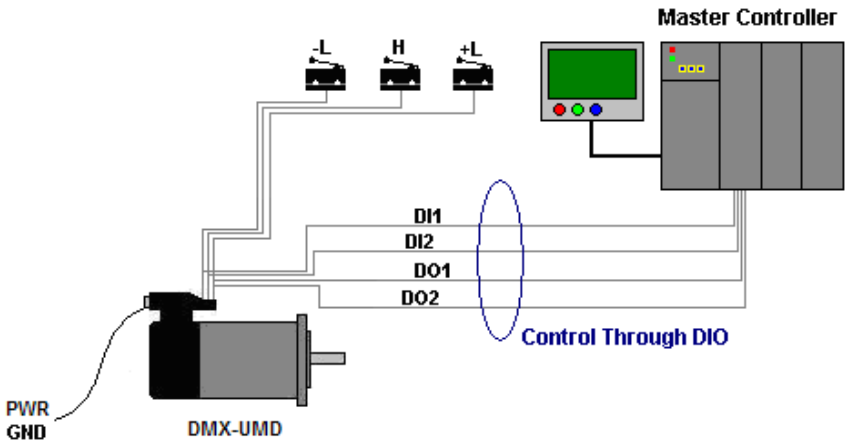


Figure 6.3

DIO communication allows the user to store 16 different types (see section 6.5.3) of moves into DMX-UMD flash memory. These moves can be referenced using the select bits (**DI3-DI6**) and triggered by using the start bit (**DI1**). Motion can be aborted by triggering the abort/clear bit (**DI2**). If an error occurs, it can also be cleared by triggering the abort/clear bit (**DI2**).

6.5.2. DIO Latency

Digital input response time to a trigger from start bit **(DI1)** is about 10 micro seconds. The actual amount of time from trigger to the beginning of the motion move depends on the command.

6.5.3. Setting Up DIO Parameters

In order to use this feature, you must first enable DIO mode (using **EDIO** command) as well as configure the appropriate DIO parameters via USB.

The DIO parameters are set using the **MP[X][Y]** command.

To view parameters, use command **MP[X][Y]**. To set values, use **MPXY=[value].**

X Parameter:
This parameter corresponds to the 2^4=16 selections that can be selected by DI3-DI6. This character must be written in hexadecimal (i.e. 0-F).

Y Parameter:
This parameter corresponds to the 5 different values that correspond to each DIO move. See the table below.

Note that some move operations do not need all 5 parameters. In this case, any extra move values that are entered will be ignored. For example, the STOP command does not need a “Target Position”. Any value entered here will be ignored in this case.

Y Parameter

Y	Description
0	DIO Move reference (see Table 10.1)
1	Target Position
2	Low Speed
3	Acceleration
4	High Speed

Table 6.2

DIO Move List

Move Reference	Command
0	None
1	STOP
2	X[Target Position]
3	INC+ [Current Position + Target Position]
4	INC- [Current Position - Target Position]
5	J+
6	J-
7	H+
8	H-
9	EO=0
10	EO=1
11	ZH+
12	ZH-
13	SSPD[High Speed]
14	SCV=1
15	SCV=0
16	SL=1
17	SL=0
18	PX=[Target Position]
19	EX=[Target Position]
20	Z+
21	Z-
22	SSPDM=[High Speed]

Table 6.3

6.5.4. Examples

1. **Make DIO selection "0" correspond to the J+ command with the following parameters:**

   Target Position = NA
   Low Speed = 100
   Acceleration = 300
   High Speed = 1000

   Send commands:

```
MP00 = 5        ` Set move reference for "0" to J+
MP01 = 0        ` Set target position to 0 (value will be ignored)
MP02 = 100      ` Set low speed to 100
MP03 = 300      ` Set acceleration to 300
MP04 = 1000     ` Set high speed to 1000
```

2. **Make DIO selection "0xF" correspond to the X800 command with the following parameters:**

   Target Position = 800
   Low Speed = 500
   Acceleration = 500
   High Speed = 5000

   Send commands:

```
MPF0 = 2          ` Set move reference for "F" to X[value]
MPF1 = 800        ` Set target position to 800
MPF2 = 500        ` Set low speed to 500
MPF3 = 500        ` Set acceleration to 500
MPF4 = 5000       ` Set high speed to 5000
```

6.5.5. Using DIO

1. First drive the **select bits** (**DI3-DI6).**
2. Then pull **start bit** (**DI1)** low to begin the move. (falling-edge triggered)
3. Trigger **abort/clear bit** (**DI2**) to abort motion command if desired.

Figure 6.4 shows a timing diagram using DIO control.

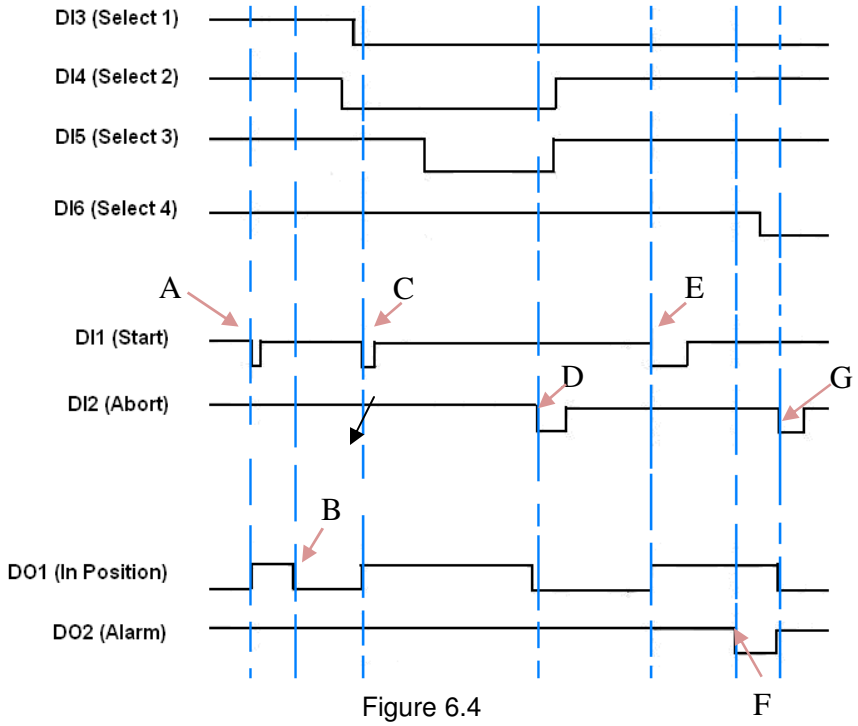


Figure 6.4

A) On falling edge of **Start**, motion command stored in memory location 0 (0000) is triggered. **In Position** turns off.

B) After motion command 0 (0000) is complete, **In Position** turns on.

C) On falling edge of **Start**, motion command stored in memory location 12 (1100) is triggered. **In Position** turns off.

D) On falling edge of **Abort**, motion stops immediately. **In Position** turns on. Note: If move was an absolute move type, and target position was not reached, **In Position** will instead remain off.

E) On falling edge of **Start**, motion command stored in memory location 8 (1000) is triggered. **In Position** turns off.

F) Motion error occurs (i.e. limit error or StepNLoop error). **Alarm** turns on. **In Position** stays off. Controller is now in error state.

G) On falling edge of **Abort**, error state is cleared. **In Position** turns on.

DIO communication is not allowed while a standalone programming is running. If DIO communication is enabled while a standalone program begins execution, it will be automatically disabled.

Triggering the **start bit (DI1)** will not trigger a motion move if the **abort bit (DI2)** is on, or if the controller is in error state. If the controller is in error state, first clear the error by triggering the **abort/clear bit (DI2)**.

The alarm bit output is on whenever there is either a SNL or limit error.

The in position bit output is on whenever the motor is in position.

Signals are active low.

7. 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 using when writing a standalone program onto the DMX-UMD.

7.1. Motion Profile and Speed

By default, a trapezoidal velocity profile is used. See Figure 7.0.

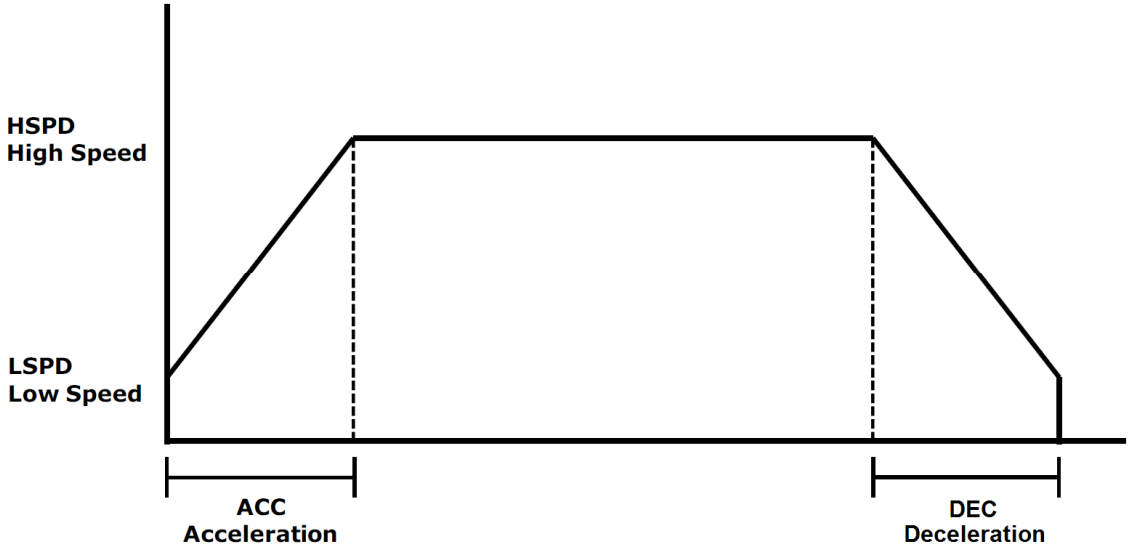


Figure 7.0

S-curve velocity profile can also be achieved by using the **SCV=1** command, as shown in Figure 7.1.

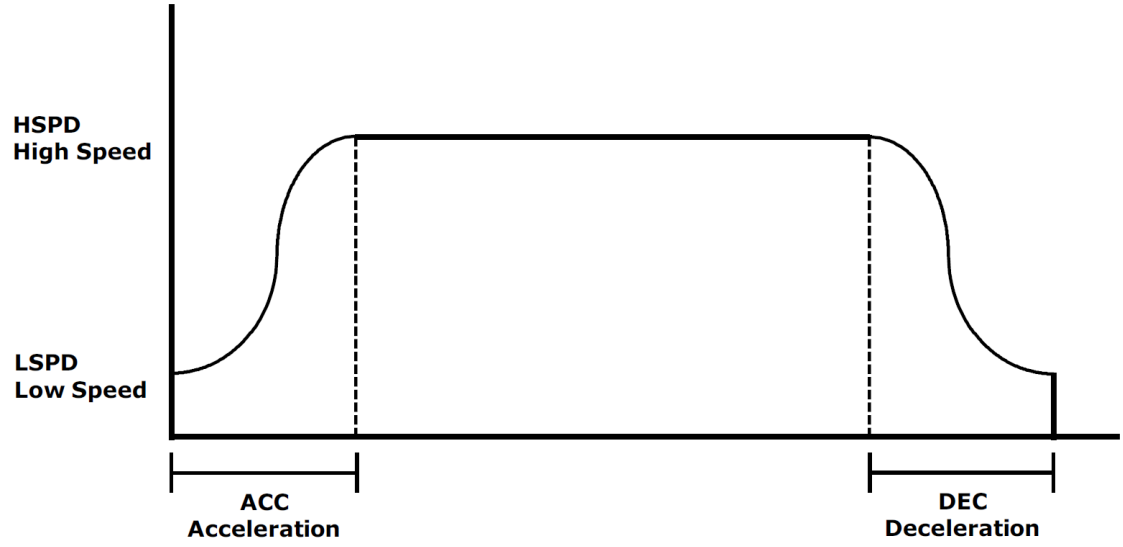


Figure 7.1

Once a typical move is issued, the motor 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 **HSPD** and **LSPD** commands to modify the high speed and low speed settings. Depending on the voltage, current, motor type, and acceleration value, the maximum achievable speed will vary.

Acceleration and deceleration time is in milliseconds. Use the **ACC** command to access the acceleration setting and the **DEC** command to access the deceleration setting. By default, the acceleration setting will be used for both the acceleration and deceleration in the motion profile. In order to decelerate using the value set in the **DEC** parameter, set the **EDEC** setting to 1.

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

ASCII	**HSPD**	**LSPD**	**ACC**	**DEC**	**EDEC**	**SCV**
Standalone	**HSPD**	**LSPD**	**ACC**	**DEC**	**-**	**SCVX**

7.2. On-The-Fly Speed Change

An on-the-fly speed change can be achieved at any point while the motor is in motion. In order to perform an on-the-fly speed change, s-curve velocity profile must be disabled.

Before an on-the-fly speed change is performed, the correct speed window must be selected. To select a speed window, use the ASCII command **SSPDM** or the standalone command **SSPDMX**. Choosing the correct speed window will depend on the initial target speed and the final target speed. Both speeds will need to be within the same speed window.

The speed window must be set while the motor is idle. Refer to Appendix A for details on the speed windows.

Once the speed window has been set, an on-the-fly speed change can occur anytime the motor is in motion. The ASCII command **SSPD=[speed]** or the standalone command **SSPDX=[speed]** can be used to perform the actual speed change.

For non on-the-fly speed change moves, set the speed window to 0.

ASCII	**SSPD**	**SSPDM**
Standalone	**SSPDX**	**SSPDMX**

7.3. Position Counter

DMX-UMD has 32 bit signed step position counter. Range of the position counter is from –2,147,483,648 to 2,147,483,647. Get the current step position by using the **PX** command.

The **PX** command can also be used to manually set the position of the motor. If the motor is moving while an attempt is made to set the position, an error will be returned and the position will remain unchanged.

Similarly, the DMX-UMD also has a 32 bit signed encoder position counter. The built in encoder will have a resolution of 1000 counts/revolution. With quadrature decoding, the resolution is increased to 4000 counts/revolution. Use the **EX** command to read and set the encoder position.

When StepNLoop closed-loop control is enabled, the **EX** command returns the encoder position and the **PX** command returns the real-time target position of the motor.

When StepNLoop closed-loop control is disabled, the **EX** command returns the encoder position and the **PX** command returns the step position. See section 7.16 for details on the StepNLoop feature.

ASCII	**PX**	**EX**
Standalone	**PX**	**EX**

7.4. Motor Power

Using the **EO** command, the motor power can be enabled or disabled. By default, the enable output is turned off at boot-up.

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.

ASCII	**EO**	**EOBOOT**
Standalone	**EO**	**-**

7.5. Jog Move

A jog move is used to continuously move the motor without stopping. Use the **J+/J-** command when operating in ASCII mode and the **JOGX+/JOGX-** in 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 9.1 for details on error responses.

ASCII	**J[+/-]**
Standalone	**JOGX[+/-]**

7.6. Stopping the Motor

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. To stop abruptly, use the **ABORT** command in ASCII mode and **ABORTX** in standalone. The ASCII command **STOP,** and standalone command **STOPX**, can be used to stop the motor with deceleration.

ASCII	**STOP**	**ABORT**
Standalone	**STOPX**	**ABORTX**

7.7. Positional Moves

Positional moves are used to move the motor to a desired position. The **X[target]** command should be used make positional moves.

When StepNLoop is enabled, the target position in positional moves will be in units of encoder counts. When StepNLoop is disabled, the target position will be in units of motor steps. See section 7.16 for details on the StepNLoop feature.

The DMX-UMD also has the ability to move in an absolute or incremental mode. Absolute move mode will move the motor to the target position, while incremental move mode will increment the current position by the target position. The **INC** and **ABS** commands set the move mode. Use the **MM** command to read the current move mode. If the **MM** command returns 0, the motor is in absolute mode. A value of 1 will indicate the motor is in increment mode.

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 9.1 for details on error responses.

ASCII	**X[pos]**	**INC**	**ABS**	**MM**
Standalone	**X[pos]**	**INC**	**ABS**	**-**

7.8. On-The-Fly Target Position Change

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

If a **T** command is sent while the controller is not performing a target move, the command is not processed. Instead, an error response is returned. See table 9.1 for details on error responses.

ASCII	**T[pos]**
Standalone	**-**

7.9. Homing

Home search routines involve moving the motor and using the home, limit, or Z-index inputs to determine the zero reference position. Five different types of homing routines are available.

The homing routines that involve a decelerated stop will result in a final position that is non-zero. In this case the zero reference position will be the position where the deceleration occurred. The ASCII command **RZ=1** can be used to perform an automated return to the zero reference position after the deceleration is complete.

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 9.1 for details on error responses.

7.9.1. Home Input Only (High Speed Only)

Use the **H+/-** command for ASCII mode or the **HOMEX+/-** command for standalone mode. Figure 7.2 shows the homing routine.

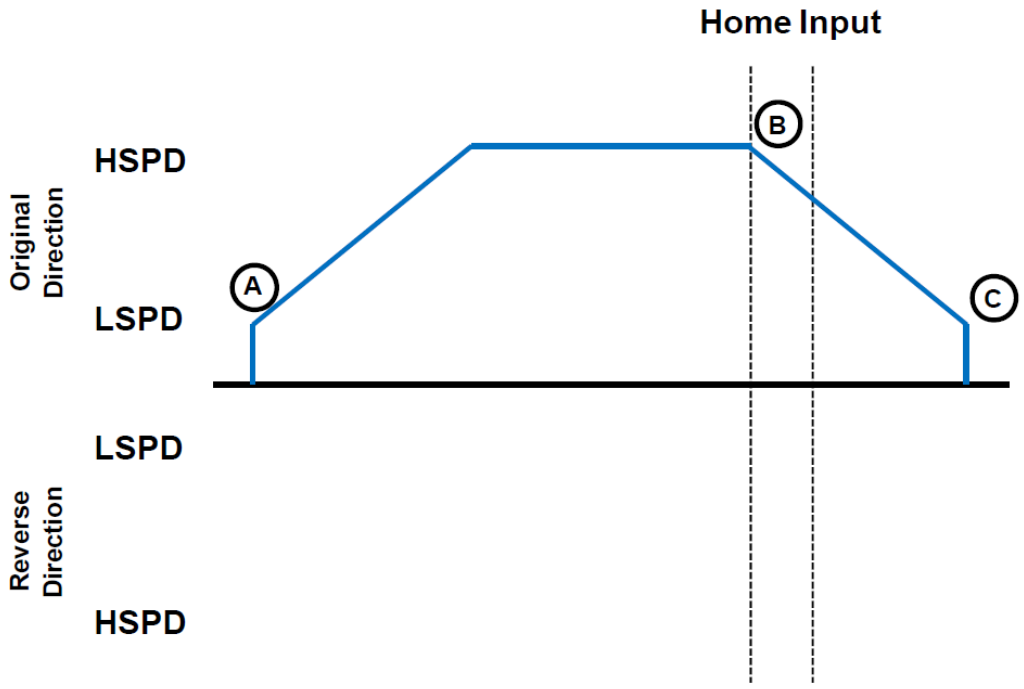


Figure 7.2

A. 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 begins to decelerate to low speed. As the motor

decelerates, the position counter keeps counting with reference to the zero position.

C. Once low speed is reached, the motor stops. The position is non-zero however the zero position is maintained. If **RZ=1**, the motor will return to its actual zero position.

ASCII	**H+/-**	**RZ**
Standalone	**HOMEX+/-**	**-**

7.9.2. Home Input and Z-index

Use the **ZH+/-** command for ASCII mode or the **ZHOMEX+/-** command for standalone mode. Figure 7.3 shows the homing routine.

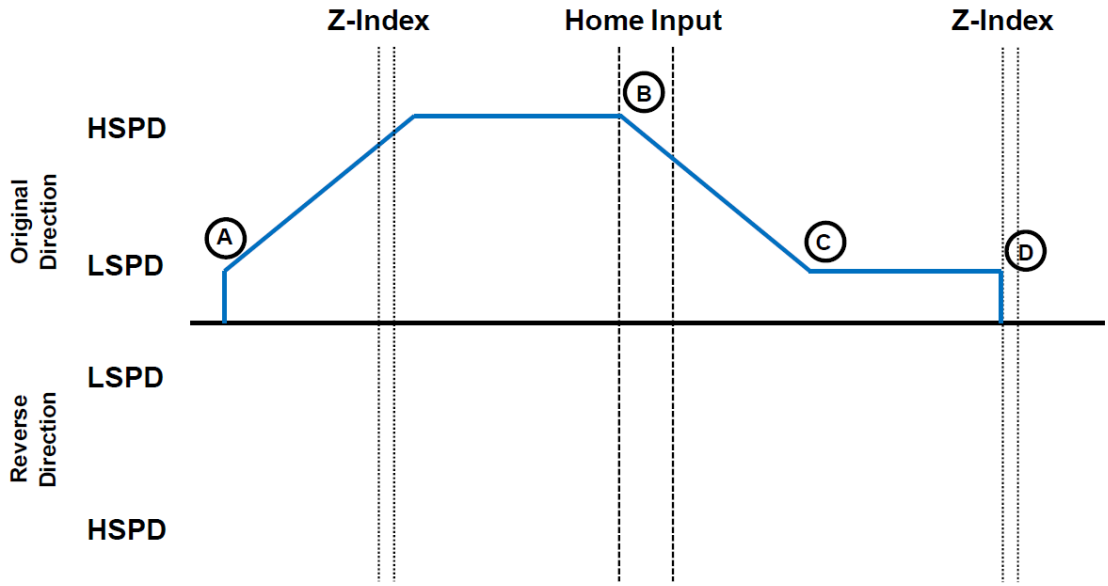


Figure 7.3

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 motor decelerates to low speed
C. Once low speed is reached, the motor begins to search for the z-index pulse.
D. Once the z-index pulse is found, the motor stops and the position is set to zero.

ASCII	**ZH+/-**
Standalone	**ZHOME+/-**

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

Use the **HL+/-** command for ASCII mode or the **HLHOMEX+/-** command for standalone mode. Figure 7.4 shows the homing routine.

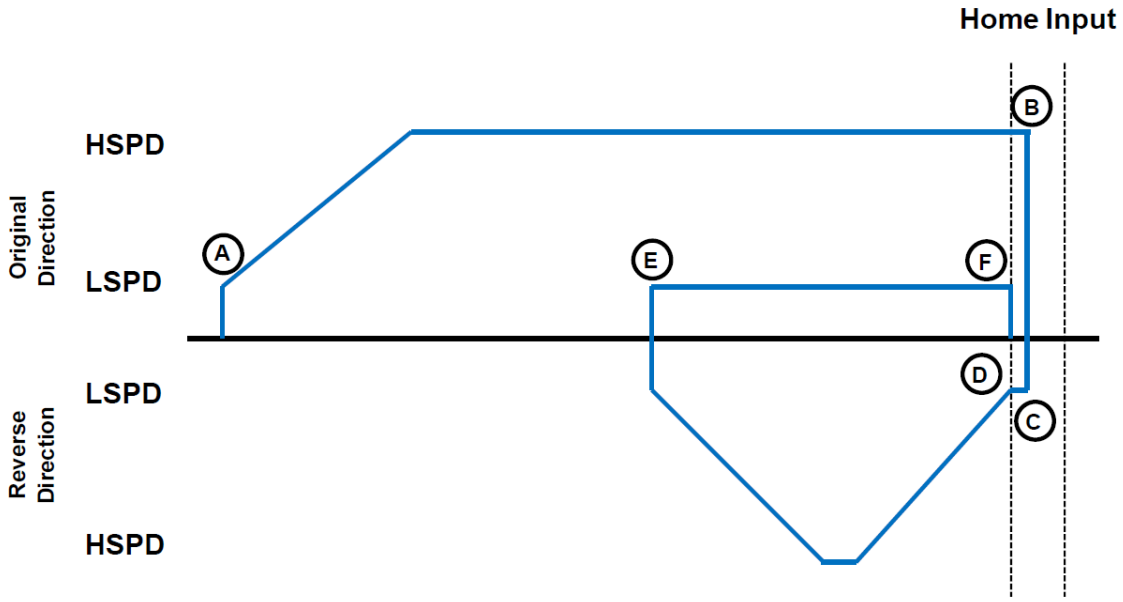


Figure 7.4

A. 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 immediately stops.
C. The motor moves at low speed in the reverse direction until the home input has been cleared.
D. Once the home input is cleared, the motor will continue to move in the reverse direction by the amount defined by the home correction amount (**HCA**). It will ramp up to high speed for this movement.
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.

ASCII	**HL+/-**	**HCA**
Standalone	**HLHOMEX+/-**	**-**

7.9.4. Limit Only

Use the **L+/-** command in ASCII mode or the **LHOMEX+/-** command for standalone mode. Figure 7.5 shows the homing routine.

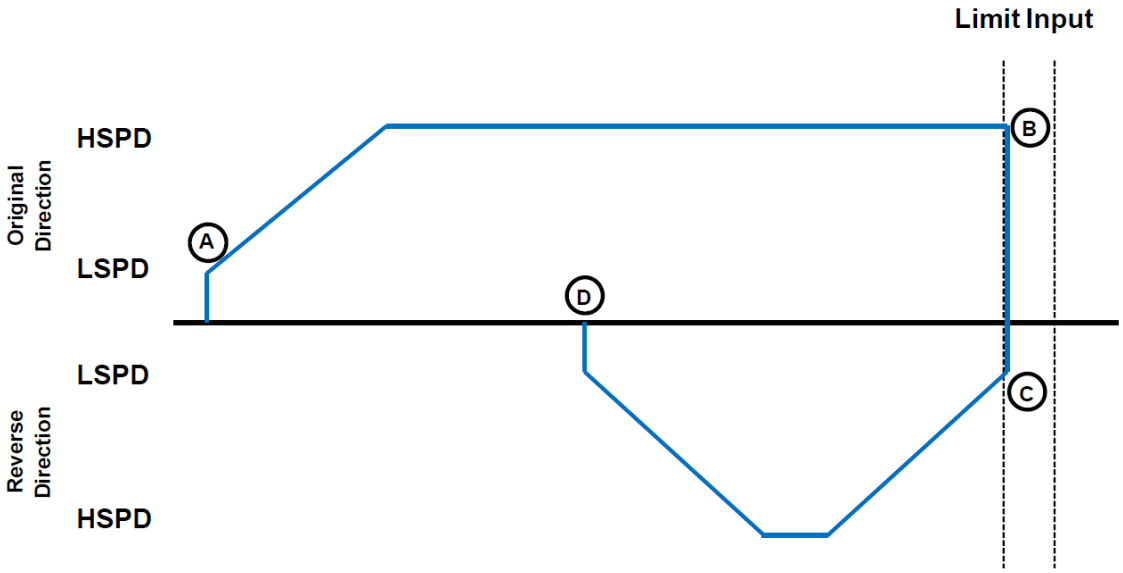


Figure 7.5

A. 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.
C. The motor position will be set to the limit correction amount (**LCA**). It will the move in the reverse direction at high speed.
D. Once the limit correction amount move is complete, the motor position will read zero.

ASCII	**L+/-**	**LCA**
Standalone	**LHOMEX+/-**	**-**

7.9.5. Z-index Only

Use the **Z+/-** command for ASCII mode or the **ZOMEX+/-** command for standalone mode. Figure 7.6 shows the homing routine.

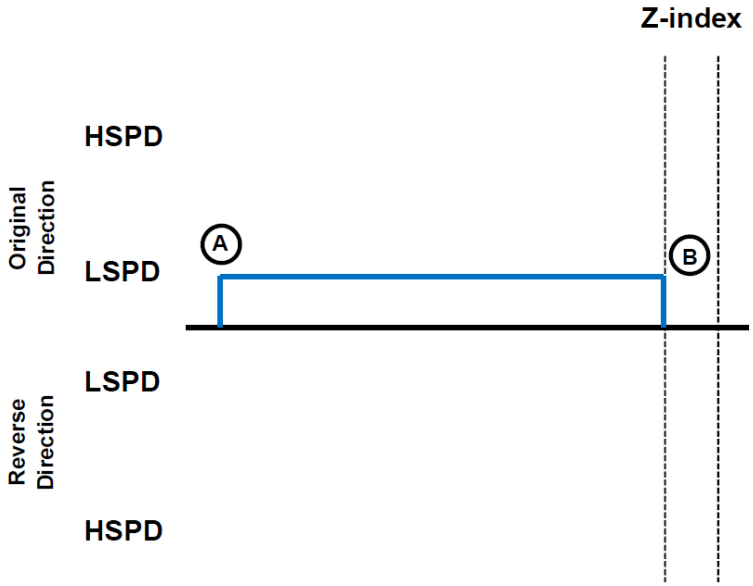


Figure 7.6

A. Issuing the command starts the motor at low speed.
B. Once the z-index pulse is found, the motor stops and the position is set to zero.

ASCII	**Z+/-**
Standalone	**ZOMEX+/-**

7.10. Limit Switch Function

With the limit switch function enabled, triggering of the limit switch while the motor is moving will stop the motion immediately. For example, if the positive limit switch is triggered while moving in the 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.

Once the limit error is set, the status error must be cleared, using the **CLR** command in ASCII mode or the **ECLEARX** command in the standalone mode, in order to move the motor again.

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

ASCII	**CLR**	**IERR**
Standalone	**ECLEARX**	**-**

7.11. Motor Status

Motor status can be read anytime using the **MST** command. Table 7.0 shows the bit representation for motor status.

Bit	Description
0	Motor running at constant speed
1	Motor in acceleration
2	Motor in deceleration
3	Home input switch status
4	Minus limit input switch status
5	Plus limit input switch status
6	Minus limit error. This bit is latched when minus limit is hit during motion. This error must be cleared using the **CLR** command before issuing any subsequent move commands.
7	Plus limit error. This bit is latched when plus limit is hit during motion. This error must be cleared using the **CLR** command before issuing any subsequent move commands.
8	Latch input status
9	Z-index status
10	TOC time-out status

Table 7.0

ASCII	**MST**
Standalone	**MSTX**

7.12. Digital Inputs/Outputs

DMX-UMD module comes with 6 digital inputs and 2 digital outputs, which can be used for general purpose or DIO control. Enable/disable DIO control mode by using the **EDIO** command.

7.12.1. Digital Inputs

The digital input status of all 6 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[1-6]** commands. Note that the indexes are 1-based for the bit references. For example, DI1 refers to bit 0, not bit 1. See Table 7.1 for details.

Bit	Description	Bit-Wise Command
0	Digital Input 1 (Start)	DI1
1	Digital Input 2 (Abort/Clear)	DI2
2	Digital Input 3 (Select 1)	DI3
3	Digital Input 4 (Select 2)	DI4
4	Digital Input 5 (Select 3)	DI5
5	Digital Input 6 (Select 4)	DI6

Table 7.1

ASCII	**DI**	**DI[1-6]**	**EDIO**
Standalone	**DI**	**DI[1-6]**	**-**

7.12.2. Digital Outputs

When DIO control is disabled, you can drive DO1 and DO2 by using the **DO** command. DO value must be within the range of 0-3.

Digital output values can also be referenced one bit at a time by the **DO[1-2]** commands. Note that the indexes are 1-based for the bit references. For example, DO1 refers to bit 0, not bit 1. See Table 7.2 for details.

Bit	Description	Bit-Wise Commands
0	Digital Output 1 (In Position)	DO1
1	Digital Output 2 (Alarm)	DO2

Table 7.2

If StepNLoop control and **EDIO** are enabled, DO1 is used as an “In Position” status output, and DO2 is used as an “Alarm” output. See section 7.16 for details on StepNLoop control.

If 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 7.15 for details. Digital outputs are active low by default.

The initial state of both 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.

ASCII	**DO**	**DO[1-2]**	**DOBOOT**
Standalone	**DO**	**DO[1-2]**	**-**

7.13. High Speed Latch Input

The DMX-UMD module provides a high speed position latch input.

This input performs high speed position capture of both pulse and encoder positions but does not reset the pulse or encoder position counters.

When StepNLoop mode is enabled, the position value will be the current target position of the motor.

Use the ASCII command **LT** or the standalone command **LTX** to enable and disable latch feature. To read the latch status, use the **LTS** ASCII command or the **LTSX** standalone command. Table 7.3. details the value representation of the latch status.

Return Value	Description
0	Latch off
1	Latch on and waiting for latch trigger

2	Latch triggered

Table 7.3

Once the latch is triggered, the triggered positions can be retrieved using the **LTP** ASCII command or the **LTPX** standalone command (latched pulse position) and the **LTE** ASCII command or the **LTEX** standalone command (latched encoder position) commands.

ASCII	**LT**	**LTS**	**LTP**	**LTE**
Standalone	**LTX**	**LTSX**	**LTPX**	**LTEX**

7.14. Sync Output

DMX-UMD has a designated synchronization digital output (DO2). The synchronization output is triggered when the encoder position value meets a defined condition. While this feature is enabled, digital output 2 cannot be controlled by user.

Use the ASCII command **SYNO** or the standalone command **SYNONX** to enable the synchronization output feature. Similarly the ASCII command **SYNF** or the standalone command **SYNOFFX** can be used to disable the synchronization output feature.

Use **SYNP** ASCII command or the **SYNPOSX** standalone command to read and set the synchronization position value. The synchronization output feature will use this position and the synchronization condition defined by the **SYNC** ASCII command or **SYNCFGX** standalone command to determine the output status. See table 7.4. for the available synchronization conditions.

Value	Description
1	Encoder position is equal to the sync position.
2	Encoder position is less than the sync position
3	Encoder position is greater than the sync position.

Table 7.4

Once the synchronization condition is met for options [1, 2, 3], the synchronization status will be set. The **SYNS** command in ASCII mode or the **SYNSTATX** in standalone mode can be used to query the status. Table 7.5 shows the return values of the synchronization status.

Value	Description
0	Sync output feature is off
1	Waiting for sync condition

2	Sync condition has occurred

Table 7.5

Use **SYNT** to set the pulse width output time (ms). This parameter is only used if the synchronization condition is set to 1. Note the maximum pulse width is 10 ms. If this parameter is set to 0, the output pulse will depend on how long the encoder value is equal to the sync position.

ASCII	**SYNO**	**SYNF**	**SYNP**	**SYNC**	**SYNT**	**SYNS**
Standalone	**SYNONX**	**SYNOFFX**	**SYNPOSX**	**SYNCFGX**	**SYNTIMEX**	**SYNSTATX**

7.15. Polarity

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

Bit	Description	
0	Reserved	
1	Direction	
2	Reserved	
3	Reserved	
4	Limit	
5	Home	
6	Latch	
7	Z-channel index	
8,9	Encoder decoding	
	00	1X
	01	2X
	10	4X
10	Digital Output	
11	Digital Input	
12	Jump to line 0 on error	
13	Enable Output	

Table 7.6

The jump to line 0 polarity option defined by bit 11 indicates the return line once a standalone program has recovered from an error state. If this bit is low, the standalone program will return to the last processed line. If this bit is high, then it will return to the first line of the program.

All other polarity options indicate whether the input or output is active high or active low.

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

7.16. StepNLoop Closed Loop Control

DMX-UMD features a closed-loop position verification algorithm called StepNLoop (SNL). The algorithm requires the use of an incremental encoder that is included in a standard DMX-UMD.

SNL performs the following operations:

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

See table 7.7 for a list of the SNL control parameters.

SNL Parameter	Description	Command
StepNLoop Ratio	[1]Ratio between motor pulses and encoder counts. This ratio will depend on the motor type, micro-stepping, encoder resolution and decoding multiplier. Value must be in the range [0.001 , 999.999].	**SLR**
Tolerance	Maximum error between target and actual position that is considered "In Position". In this case, no correction is performed. Units are in encoder counts.	**SLT**
Error Range	Maximum error between target and actual position that is not considered a serious error. If the error exceeds this value, the motor will stop immediately and go into an error state.	**SLE**
Correction Attempt	Maximum number of correction tries that the controller will attempt before stopping and going into an error state.	**SLA**

Table 7.7

[1]A convenient way to find the StepNLoop ratio is to set EX=0, PX=0 and move the motor +1000 pulses. The ratio can be calculated by dividing 1000 by the resulting EX value. Note that the value must be positive. If it is not, then the direction polarity must be adjusted. See table 7.6 for details.

To enable/disable the StepNLoop feature use the **SL** ASCII command or the **SLX** standalone command. To read the StepNLoop status, use **SLS** ASCII command or the **SLSX** standalone command. See table 7.8 for a list of the StepNLoop status return values.

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

Table 7.8

Depending on the value of the delta, the StepNloop algorithm can have certain behaviors. See table 7.9 for StepNLoop behavior within different scenarios.

Condition	SNL behavior (motor is moving)	SNL behavior (motor is idle)
δ <= SLT	Continue to monitor the **DX**	In Position. No correction is performed.
δ > SLT AND δ < SLE	Continue to monitor the **DX**	Out of Position. A correction is performed.
δ > SLT AND δ > SLE	Stall Error. Motor stops and signals and error.	Error Range Error. Motor stops and signals and error.
Correction Attempt > SLA	NA	Max Attempt Error. Motor stops and signals and error.

Table 7.9

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

Once SNL is enabled, position move commands are in terms of encoder position. For example, X1000 means to move the motor to the encoder position 1000.

Once SNL is enabled, the speed is in encoder speed. For example HSPD=1000 when SNL is enabled means that the target high speed is 1000 encoder counts per second.

If DIO mode is on while SNL is enabled, DO1 is dedicated as the “In Position” output and DO2 is dedicated as the “Alarm” output. In order to use the digital outputs for general purpose, disable DIO by setting **EDIO=0**.

7.17. Standalone Programming

Standalone programming allows the controller to execute a user defined program that is stored in the internal memory of the DMX-UMD. The standalone program can be run independently of serial communication or while communication is active.

Standalone programs can be written to the DMX-UMD using the Windows GUI described in section 8. Once a standalone program is written by the user, it is then compiled and downloaded to the DMX-UMD. Each line of written standalone code creates 1-4 assembly lines of code after compilation

The DMX-UMD has the ability to store and operate two separate standalone programs simultaneously.

7.17.1. 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.

7.17.2. Standalone Control

The DMX-UMD supports the simultaneous execution of two standalone programs. Program 0 is controlled via the **SR0** command and program 1 is controlled via the **SR1** command. For examples of multi-threading, please refer to section 10. The following assignments can be used to control a standalone program.

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

Table 7.10

7.17.3. Standalone Status

The **SASTAT[0-1]** command can be used to determine the current status of the specified standalone program. Table 7.11 details the return values of this command.

Value	Description
0	Idle
1	Running
2	Paused

3	N/A
4	Errored

Table 7.11

The **SPC[0-1]** 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[0-1]** command is referencing the assembly language line of code that does not directly transfer to the pre-compiled user generated code. The return value can range from [0-1784].

7.17.4. Standalone Subroutines

The DMX-UMD 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 subroutine can be shared by both standalone programs. Refer to section 10 on 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.

7.17.5. Error Handling

Subroutine 31 is designated for error handling. If an error occurs during standalone execution (i.e. limit error, StepNLoop 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. Typically the code within subroutine 31 will contain the standalone command **ECLEARX** in order to clear the current error. Section 10 will contain examples of using subroutine 31 to perform error handling.

The return jump from subroutine 31 will be determined by bit 12 of the **POL** register. This setting will determine if the standalone program will jump back to the beginning of the program or to the last performed line. Refer to table 7.6 for details.

7.17.6. Standalone Variables

The DMX-UMD has 100 32-bit signed standalone variables available for general purpose use. They can be used to perform basic calculations and support integer operations. The **V[0-99]** command can be used to access the specified variables. The syntax for all available operations can be found 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 7.12

Variables V50 through V99 can be stored to flash memory using the **STORE** command. Variables V0-V49 will be initialized to zero on power up.

7.17.7. 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 save the setting to flash memory. It will take effect on the following power cycle. See description in Table 7.13 for the bit assignment of the **SLOAD** setting.

Bit	Description
0	Standalone Program 0
1	Standalone Program 1

Table 7.13

Standalone programs can also be configured to run on boot-up using the Windows GUI. See section 8 for details.

ASCII	**SR[0-1]**	**SASTAT[0-1]**	**SPC[0-1]**	**GS[0-31]**	**V[0-99]**	**SLOAD**
Standalone	**SR[0-1]**	**-**	**-**	**GOSUB[0-31]**	**V[0-99]**	**-**

7.18. Microstep Driver Configuration

The built in driver of DMX-UMD can be configured via software. See below for commands relating to driver configuration.

Command	Description
DRVMS	Micro-stepping value of the driver [2-500].
DRVRC	Run current value of the driver [100-3000 mA] (peak current).
DRVIC	Idle current value of the driver [100-2800 mA] (peak current).
DRVIT	Idle time value of the driver [1-100 centi-sec]. This is the amount of time the driver waits before dropping from the run current to idle current value.
RR	Get the driver parameters. DRVMS/DRVRC/DRVIC/DRVIT values will not

	be valid until the controller reads the driver parameters by issuing the RR command. Once this command is issued, communication to DMX-UMD will not be available for 2 seconds.
R2	Get the read operation status. After issuing the RR command and waiting 2 seconds, get the read operation status by using the R2 command. A return value of 1 signifies a successful read. All other return values signify a failed read operation.
RW	Write driver parameters. After DRVMS/DRVRC/DRVIC/DRVIT parameters are set by the user, they are not actually written to the driver until the RW command is sent. Once this command is issued, communication to DMX-UMD will not be available for 2 seconds.
R4	Get the write operation status. After issuing the RW command and waiting 2 seconds, get the write operation status by using the R4 command. A return value of 1 signifies a successful write. All other return values signify a failed write operation.
DRVMS	Micro-stepping value of the driver [2-500].

Table 7.14

Driver configuration can also be done via standalone code.
While reading or writing to the micro-step driver, StepNLoop, joystick control and DIO control modes must be disabled. These control modes may interfere with the driver configuration.

7.19. Storing to Flash

The following items are stored to flash. To store to flash, use the **STORE** command.

ASCII Command	Description
DB	Serial communication baud rate
DN	Device name
DNM	$_1$Modbus device number
DOBOOT	DO configuration at boot-up
DRVMS, DRVRC, DRVIC, DRVIT	Micro-step driver settings
EDEC	Unique deceleration enable
EDIO, MP	DIO communication settings
EOBOOT	EO configuration at boot-up
HCA	Home correction amount
IERR	Ignore limit error enable
LCA	Limit correction amount
POL	Polarity settings
RSM	$_1$Modbus enable
RT	ASCII response type
RZ	Return to zero position after homing
SL, SLR, SLE, SLT,	StepNLoop parameters

SLA	
SLOAD	Standalone program run on boot-up parameter
TOC	Time-out counter reset value
V50-V99	Note that on boot-up, V0-V49 are reset to value 0

Table 7.15

[1]See “Modbus_Addition_Addendum_A” document for details.

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

7.20. Default Settings

Following are the factory default settings when then unit is shipped from the factory.

Command	Parameter Description	Value
CURI	Idle current	500 mA
CURR	Run current	1600 mA
CURT	Idle time	500 mSec
DB	Baud rate	9600
DN	Device ID	UMD01
DOBOOT	Digital output boot-up state	7
EDO	Alarm/ in position output mode	Enabled
EOBOOT	Enable output boot-up state	0
HCA	Home correction amount	1000
IERR	Ignore error state	Disabled
LCA	Limit correction amount	1000
POL (bit 1)	Direction polarity	CW
POL (bit 4)	Limit polarity	Active Low
POL (bit 5)	Home polarity	Active Low
POL (bit 6)	Latch polarity	Active Low
POL (bit 7)	In position output polarity	Active Low
POL (bit 8)	Alarm output polarity	Active Low
POL (bit 9)	Digital output polarity	Active Low
POL (bit 10)	Digital input polarity	Active Low
POL (bit 11)	Jump to line 0 on error	Disabled
POL (bit 12)	Motor enable	Active Low
RT	Response type	Do Not Append
RZ	Return to home position	Disabled

SL	StepNLoop enable	Enabled
SLA	StepNLoop maximum attempt	10
SLE	StepNLoop error range	1000
SLOAD	Run program(s) on power up	0
SLT	StepNLoop tolerance range	10
TOC	Time-out counter value (Watch-dog)	0
V0-V99	Volatile and non-volatile variables	0

Table 7.16

8. Software Overview

The DMX-UMD has a Windows compatible software that allows for USB or RS485 communication. Standalone programming, along with all other available features of the DMX-UMD, will be accessible through the software. It can be downloaded from the Arcus Technology website.

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

Startup the DMX-UMD GUI program and you will see following screen in figure 8.0.

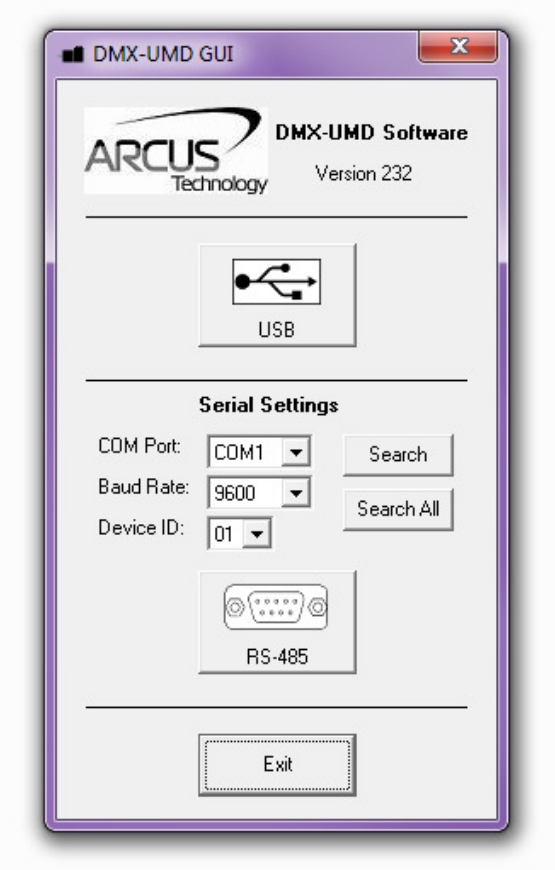


Figure 8.0

Use the USB button to select connect to a DMX-UMD over USB communication.

The first DMX-UMD connected to the PC via RS485 can be found using the Search button. If there all multiple DMX-UMD connected to the PC, the Search All button can be used to find them.

If the search fails, or you are unable to open a connection, check the following:

- Check power supply to DMX-UMD. Allowable power is range is from 12VDC to 48VDC.

- Check communication wiring. Make sure that the 485+ from DMX-UMD is connected to 485+ of the master and 485- from DMX-UMD is connected to 485- of the master.
- Confirm that the device name is set correctly. Default factory device name setting is “01”. If this name has been changed and stored to flash, enter the new name.

Once the correct serial settings have been determined, the RS-232/RS-485 button can be used to open the Main Control Screen.

8.1. Main Control Screen

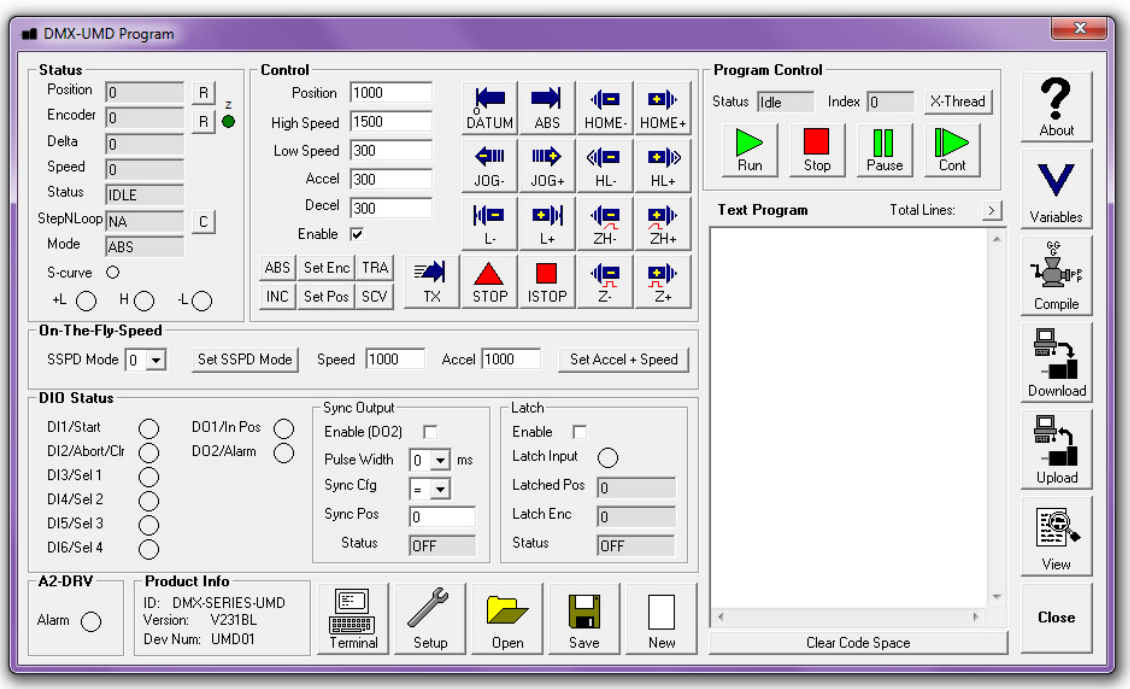


Figure 8.1

8.1.1. Status

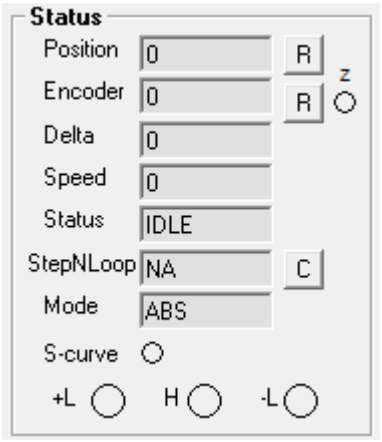


Figure 8.2

1. **Pulse Counter** – displays the current pulse position counter. When StepNLoop is enabled, this displays the Target position.
2. **Encoder Counter** – displays the current encoder position counter.
3. **Delta Counter** – valid only for StepNLoop. Displays the difference between the pulse position and the encoder position.
4. **Speed** – displays the current pulse speed output rate. Value is in pulses/second. While the controller is in StepNLoop mode, this value shows encoder counts/second.
5. **Motion Status** – displays current motion status by displaying one of the following status:
    - IDLE: motor is not moving
    - ACCEL: motion is in acceleration
    - DECEL: motion is in deceleration
    - CONST: motion is in constant speed
    - -LIM ERR: minus limit error
    - +LIM ERR: plus limit error
6. **StepNLoop Status** – valid only when StepNLoop is enabled and displays current StepNLoop status by displaying one of the following:
    - NA: StepLNoop is disabled
    - IDLE: motor is not moving
    - MOVING: target move is in progress
    - JOGGING: jog move is in progress
    - HOMING: homing is in progress
    - LHOMING: limit homing in progress
    - Z-HOMING: homing using Z-index channel in progress
    - ERR-STALL: StepNLoop has stalled
    - ERR-LIM: plus/minus limit error
7. **Move Mode** – displays current move mode.
    - ABS: all the move commands by X[pos] command will be absolute moves

- INC: all the move commands by X[pos] command will be increment moves.

8. **S-curve Status** – Displays whether the moves are in trapezoidal or S-curve acceleration.
9. **Limit/Home Input Status** – Limit and Home input status.

8.1.2. Control

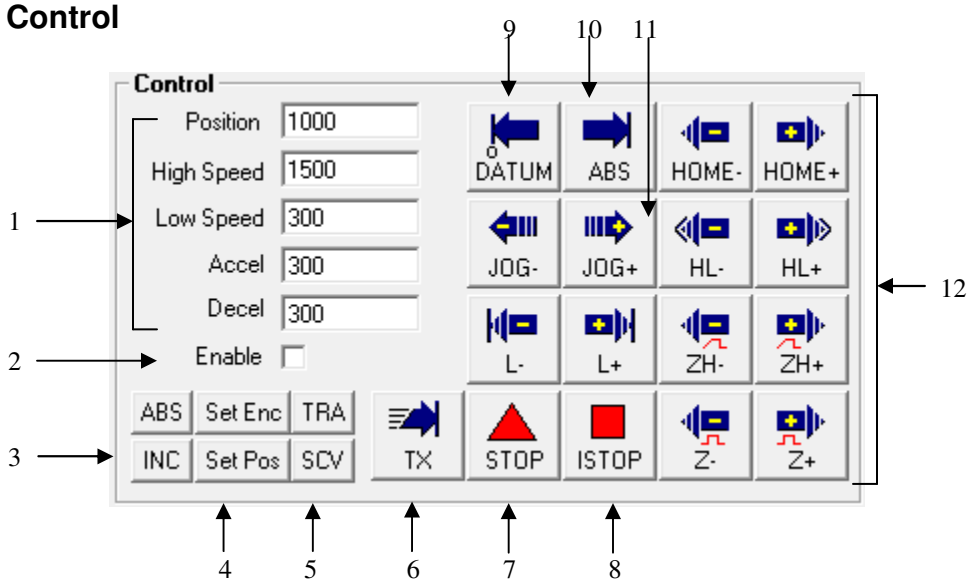


Figure 8.3

1. **Target Position/Speed/Accel**
   - Position: use this to set the target position. For normal open loop mode, this position is the pulse position and when StepNLoop is enabled this target position is in encoder position.
   - High/Low Speed: use this to set the speed of the move. For normal open loop mode, this value is in pulses/second and when StepNLoop is enabled this value is in encoder counts/second.
   - Accel: acceleration value in milliseconds.
   - Decel: deceleration value in milliseconds.
2. **Enable Driver Power** – use this button to enable and disable the power to the microstep driver.
3. **Select Move Mode** – use these buttons to select absolute or incremental move mode.
4. **Set Position** – use these buttons to set the encoder or pulse position to “Position” value.
5. **Select Acceleration Mode** – use these buttons to select trapezoidal or S-curve acceleration mode.
6. **On-the-fly target change** – Change the target position on-the-fly.
7. **Ramp Stop** – use this button to stop the motion with deceleration.

8. **Immediate Stop** – use this button to stop the motion immediately. *We recommend that ramp stop be used whenever possible to reduce the impact to the motor and the system.*
9. **Move back to zero** – use this to move the motor to the zero target position. When in absolute mode, the axis will move to zero position (zero encoder position when in StepNLoop and zero pulse position when in open loop).
10. **Perform Absolute Move** – use this to move the motor to the target position.
    When in absolute mode, the axis will move to the absolute target position.
    When in incremental mode, the axis will move incrementally.
11. **Jogging** – jog motor in either the positive or negative direction
12. **Perform Homing** – Five different homing routines are available:
    - HOME: homing is done using only the home switch.
    - HL: homing is done using only the home switch at high speed and low speed.
    - L: homing is done using the limit switch.
    - ZH: homing is done using the home switch first and then the Z index channel of the encoder.
    - Z: homing is done only using the Z index channel of the encoder.

8.1.3. On-The-Fly Speed Change

Set the speed on the fly. On-the-fly speed change feature can only be used if the controller is already in motion.

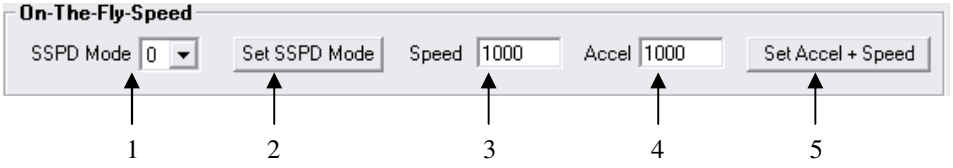


Figure 8.4

1. **On-the-fly speed mode** – Before setting the controller into motion, set the SSPDM parameter. To see which value to use, see the on-the-fly speed change section.
2. **Set SSPDM** – Set the SSPDM parameter. Note that if an on-the-fly speed change operation is to be used, this parameter must be set before the controller starts motion.
3. **Desired Speed** – Once the "Set Speed" button is clicked, the speed will change on-the-fly to the desired speed.
4. **Desired Acc/Dec** – The acceleration/deceleration use for the on-the-fly speed change operation.
5. **Set Accel + Speed** – Start the on-the-fly speed operation.

8.1.4. DIO Status

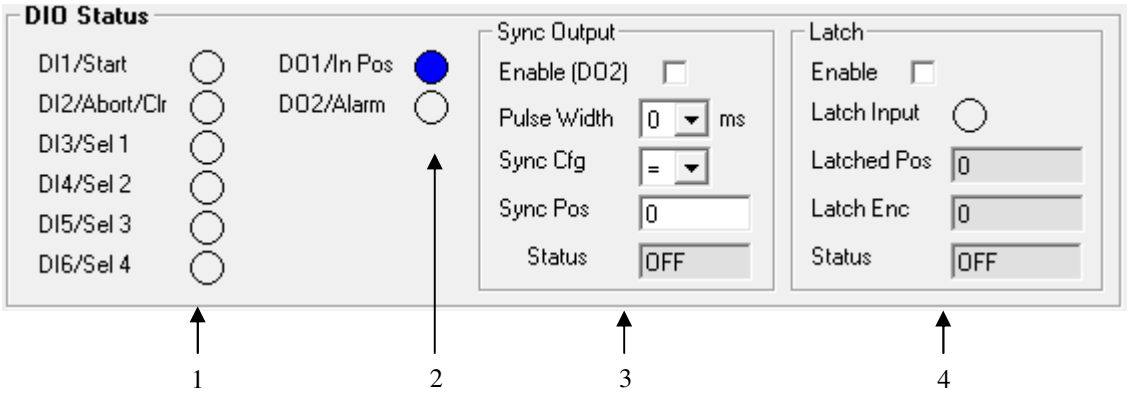


Figure 8.5

1. **Digital Input Status** – digital inputs can be used for DIO move control or as general purpose use. Refer to the setup screen to disable and enable the DIO move control.
2. **Digital Out Status and Control** – digital outs are used for StepNLoop or general purpose output use. When used as general purpose outputs, the outputs can be triggered by clicking on the circle.
3. **Sync Output** – digital outputs can be triggered.
4. **Latch** - encoder and pulse positions can be captured/latched with an input trigger.

8.1.5. DMX-A2-DRV Alarm

Figure 8.6

Status of the DMX-A2-DRV driver alarm output signal is displayed.

8.1.6. Product Info

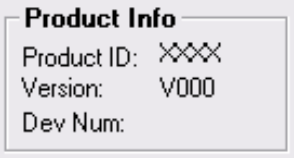


Figure 8.7

Displays the product ID as well as the firmware version.

8.1.7. Terminal

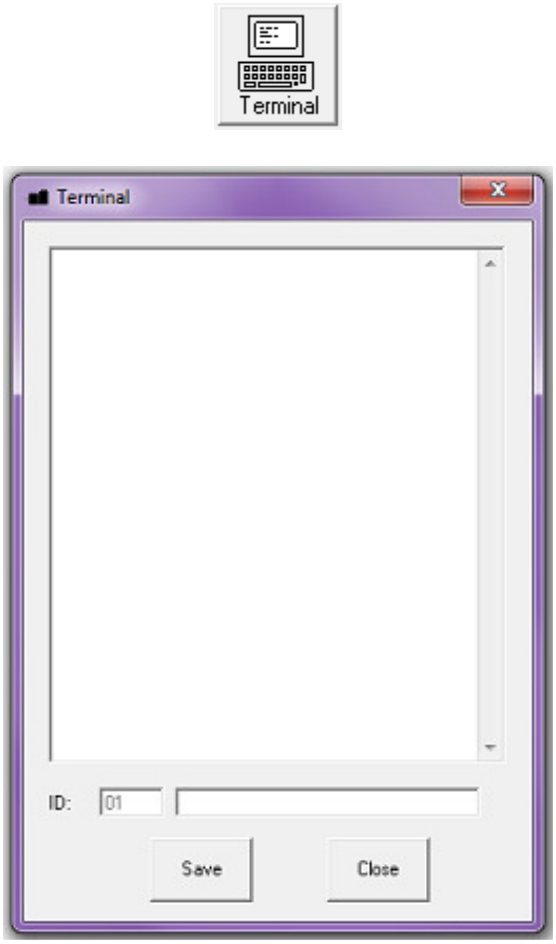


Figure 8.8

Terminal dialog box allows manual testing of the commands from a terminal screen as shown in figure 8.8

8.1.8. Setup

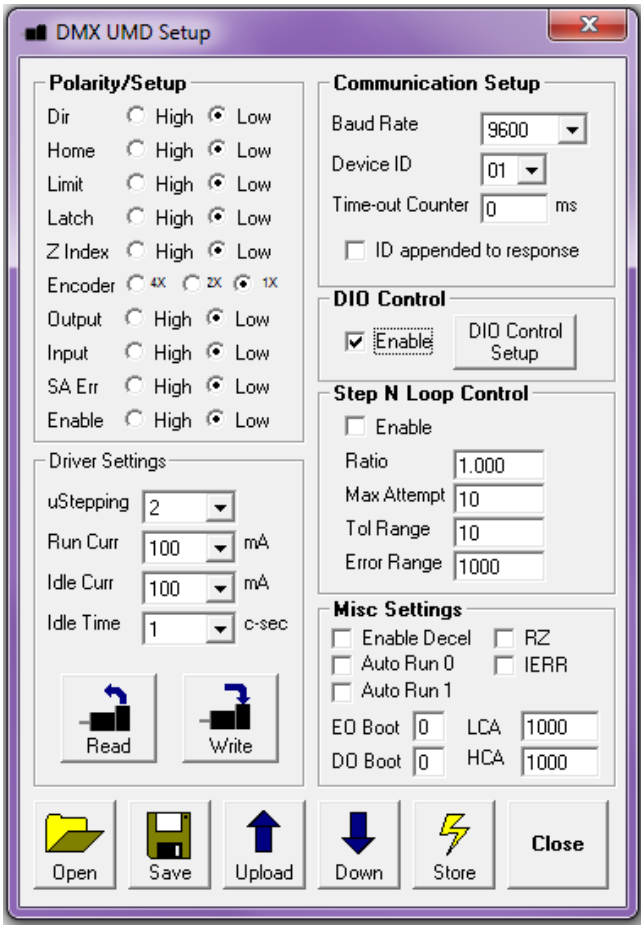


Figure 8.9

1. **Polarity Setup** – the following polarity parameters can be configured:
   - Dir: direction of the motion (clockwise or counter-clockwise)
   - Home: home input polarity
   - Limit: limit input polarity
   - Latch: latch input polarity
   - Z-Index: Encoder Z index channel polarity
   - Encoder: encoder multiplication factor can be configured as 1X, 2X, or 4X
   - Output: digital output polarity
   - Input: digital input polarity
   - SA Err: standalone error jump line:
     - Low: jump to previous line
     - High: jump to line 0

- Enable: enable output polarity

2. **Driver Setting** – The following driver settings can be configured:
   - Micro-step: 2 to 500 micro-steps
   - Run Current: 100mA to 3Amp
   - Idle Current: 100mA to 3Amp
   - Idle Time:1 to 100 centi-second (100 centi-second = 1 second)
3. **Communication Setup**
   - RS-485 communication baud rate can be selected to support different communication speed.
   - Device ID configuration allows multiple devices on the RS-485 or USB communication network.
   - Time-out counter is a watch-dog timer for communication (ms)
   - ID append to response is used by RS-485 communication for adding the device ID to any response.
4. **DIO Control** – Digital IO motion control allows motion profiles to be triggered through the digital inputs. See DIO motion control section for details. The following dialog box is shown for the DIO motion control.

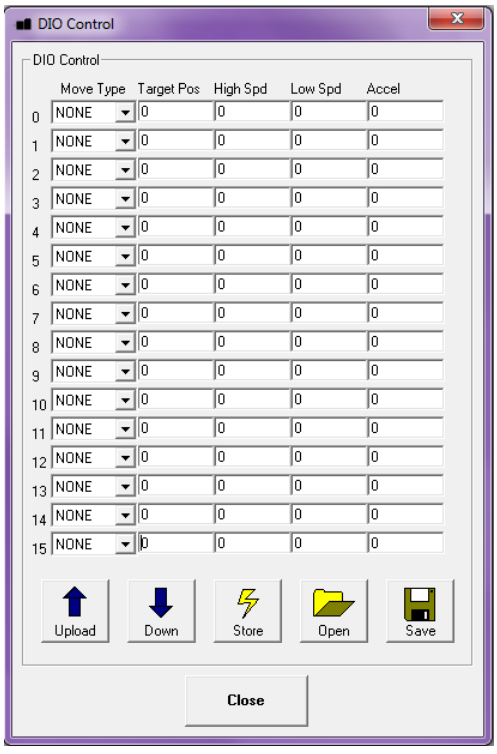


Figure 8.10

5. **StepNLoop Control** – Using the encoder input, StepNLoop control allows closed loop position verification and correction for the moves. See StepNLoop control section for details.
6. **Misc Settings**
   - Enable Decel: Allow for unique deceleration and acceleration values

- Auto Run 0: Run standalone program 0 on boot-up
- Auto Run 1: Run standalone program 1 on boot-up
- RZ: Return to zero position after homing routines
- IERR: Ignore limit error
- EOBOOT: Configure enable output boot-up state
- DOBOOT: Configure digital output boot-up state
- LCA: Set limit correction amount
- HCA: Set home correction amount

7. **Open**/**Save** – Configuration values can be saved to a file and read from a file.
8. **Upload**/**Download** – Configuration values can be uploaded and downloaded.
   Note that if the configuration values are changed, it needs to be downloaded to take effect.
9. **Store** – The downloaded parameters can be permanently stored on the non-volatile memory.

8.1.10. Standalone Program File Management

Figure 8.11

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

8.1.11. Standalone Program Editor

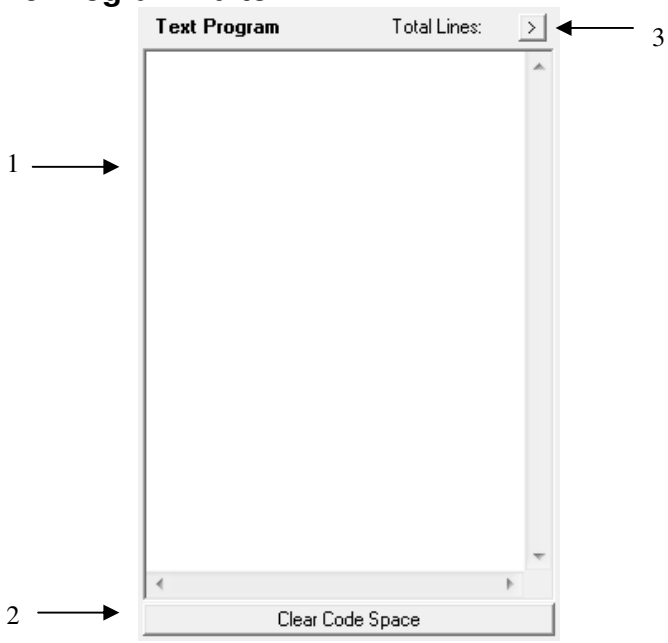


Figure 8.12

1. Write the standalone program in the Program Editor.
2. Use this button to remove the current standalone program.
3. Use this button to open a larger and easier to manage program editor.

8.1.12.Standalone Processing

Figure 8.13

1. **Compile** – Compile the standalone program.
2. **Download** – Download the compiled program.
3. **Upload –** Upload the standalone program from the controller.
4. **View** – View the low level compiled program.

8.1.13. Variable Status

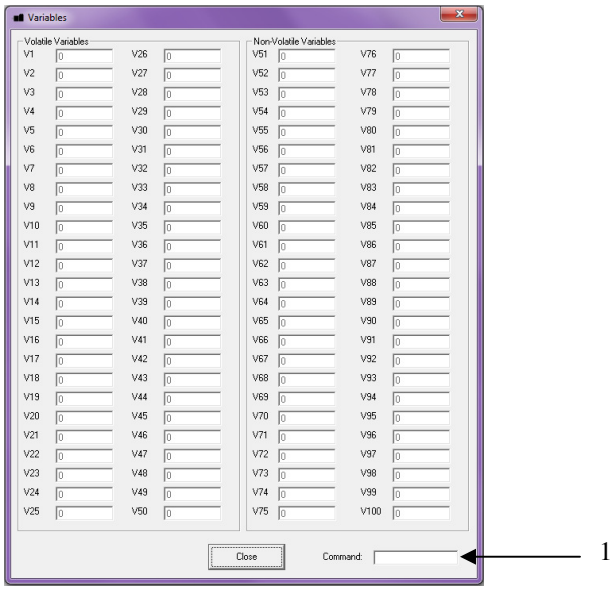


Figure 8.14

View the status of variables 1-100. Note that this window is read-only. A command line is available to send commands to the DMX-UMD.

8.1.14. Program Control

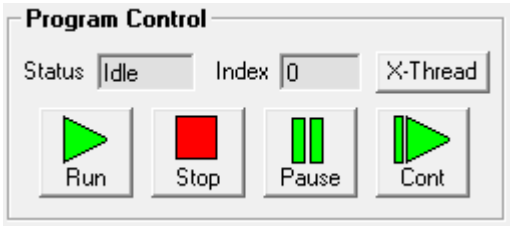


Figure 8.15

1. **Program Status** – program status shows here. Following are possible program status: Idle, Running, Errored and Paused.
2. **Index** –downloaded program is in the form of low-level code. Each line of the low level code has a line number which shows here.
3. **Run** – runs the program.
4. **Stop** – stops the program.
5. **Pause** – pauses the program.
6. **Continue** – resumes a paused program.
7. **X-Thread** – open the Program Control for standalone multi-thread operation.

9. ASCII Language Specification

Important Note: All the commands described in this section are interactive commands and are not analogous to standalone commands. Refer to section 10 for details regarding standalone commands.

DMX-UMD language is case sensitive. All command should be in capital letters. Invalid command is returned "?". Always check for proper reply when command is sent.

For USB communication, send commands identical to the ones in the following table.

For RS-485 ASCII communication, append "@XX" to the command before sending, where "XX" is the device number. Ex: To send the "J+" command to device number 05, send the following: **"@05J+"**

9.1. ASCII Command Set

Command	Description	Return
ABORT	Immediately stops the motor if in motion. For decelerate stop, use STOP command. This command can also be used to clear a StepNLoop error.	OK
ABS	Set move mode to absolute	OK
ACC	Returns current acceleration value in milliseconds.	Milli-seconds
ACC=[Value]	Sets acceleration value in milliseconds. Example: ACC=300	OK
CLR	Clears limit error as well as StepNLoop error	OK
DB	Return the current baud rate of the device	See Table 6.1
DB=[Value]	Set the baud rate of the device	OK
DEC	Get deceleration value in milliseconds. Only used if EDEC=1	Milli-seconds
DEC=[Value]	Set deceleration value in milliseconds. Only used if EDEC=1	OK
DI	Return status of digital inputs	See Table 7.1
DI[1-6]	Get individual bit status of digital inputs	0,1
DO	Return status of digital outputs	2-bit number
DO=[Value]	Set digital output 2 bit number. Digital output is writable only if DIO is disabled.	OK
DO[1-2]	Get individual bit status of digital outputs	See Table 7.2
DO[1-2]=[Value]	Set individual bit status of digital outputs	OK
DOBOOT	Get DO boot-up state	See Section 7.12.2
DOBOOT=[Value]	Set DO boot-up state	OK
DN	Get device name	[UMD01-UMD99]
DN=[Value]	Set device name	OK
DNM	Get Modbus device number	1-127
DNM=[Value]	Set Modbus device number	OK
DX	Returns the delta value during StepNLoop control	28-bit number
DRVIC	Get driver idle current setting. Value is only valid after reading parameters using the "RR" command.	[100 – 3000] mA (peak current)

DRVIC=[Value]	Set driver idle current setting. Value is only written to the driver after using the “RW” command.	OK
DRVIT	Get driver idle time setting. Value is only valid after reading parameters using the “RR” command.	[1-100] centi-sec
DRVIT=[Value]	Set driver idle time setting. Value is only written to the driver after using the “RW” command.	OK
DRVMS	Get driver micro-step setting. Value is only valid after reading parameters using the “RR” command.	[2-500] micro-stepping
DRVMS=[Value]	Set driver micro-step setting. Value is only written to the driver after using the “RW” command.	OK
DRVRC	Get driver run current setting. Value is only valid after reading parameters using the “RR” command.	[1-100] centi-sec
DRVRC=[Value]	Set driver run current setting. Value is only written to the driver after using the “RW” command.	OK
EDEC	Get unique deceleration enable	0 or 1
EDEC=[Value]	Set unique deceleration enable	OK
EDIO	Returns DIO mode status	1 – DIO enabled 0 – DIO disabled
EDIO=[0 or 1]	Enables (value 1) or disable (value 0) DIO communication	OK
EO	Returns driver power enable.	1 – Motor power enabled 0 – Motor power disabled
EO=[0 or 1]	Enables (1) or disable (0) motor power.	OK
EOBOOT	Get EO boot-up state	0 or 1
EOBOOT=[Value]	Set EO boot-up state	OK
EX	Returns current encoder counter value	28-bit number
EX=[Value]	Sets the current encoder counter value	OK
GS[0-31]	Call a subroutine that has been previously stored to flash memory	OK
HSPD	Returns High Speed Setting	PPS
HSPD=[Value]	Sets High Speed.	OK
H+	Homes the motor in positive direction	OK
H-	Homes the motor in negative direction	OK
HCA	Returns the home correction amount	28-bit number
HCA=[Value]	Sets the home correction amount	OK
HL+	Homes the motor in positive direction (with with low speed)	OK
HL-	Homes the motor in negative direction (with low speed)	OK
IERR	Get ignore limit error enable	0 or 1
IERR=[Value]	Set ignore limit error enable	OK
ID	Returns product ID	DMX-SERIES-UMD
INC	Set move mode to incremental	OK
J+	Jogs the motor in positive direction	OK
J-	Jogs the motor in negative direction	OK
L+	Limit homing in positive direction	OK
L-	Limit homing in negative direction	OK
LCA	Returns the limit correction amount	28-bit number
LCA=[Value]	Sets the limit correction amount	OK
LSPD	Returns Low Speed Setting	PPS
LSPD=[Value]	Sets Low Speed	OK
LT=[0 or 1]	Enable or disable position latch feature	OK

LTE	Returns latched encoder position	28-bit number
LTP	Returns latched pulse position	28-bit number
LTS	Returns latch status	See Table 7.3
MM	Get move mode status	0 – Absolute move mode 1 – Incremental move mode
MST	Returns motor status	See Table 7.0
MPXY	Get DIO parameter	28-bit number
MPXY=[Value]	Set DIO parameter	OK
POL	Returns current polarity	See Table 7.6
POL=[value]	Sets polarity	OK
PS	Returns current pulse speed	PPS
PX	Returns current position value	28-bit number
PX=[value]	Sets the current position value	OK
R2	Get driver read operation status	[1] – Driver read successful [2-7] – Driver read failure
R4	Get driver write operation status	[1] – Driver write successful [2-7] – Driver write failure
RR	Read driver parameters	OK
RSM	Get Modbus enable	0 or 1
RSM= [0 or 1]	Set Modbus enable	OK
RT	Get response type value	0 or 1
RT= [0 or 1]	Set response type value	OK
RZ	Get return zero enable. Used during homing	0 or 1
RZ=[0 or 1]	Set return zero enable. Used during homing	OK
RW	Write driver parameters	OK
SASTAT[0,1]	Get standalone program status 0 – Stopped 1 – Running 2 – Paused 4 – In Error	0-4
SA[LineNumber]	Get standalone line LineNumber: [0,1784]	
SA[LineNumber]=[Value]	Set standalone line LineNumber: [0,1784]	
SCV	Returns the s-curve control	0 or 1
SCV=[0 or 1]	Enable or disable s-curve. If disabled, trapezoidal acceleration/ deceleration will be used.	OK
SL	Returns StepNLoop enable status	0 – StepNLoop Off 1 – StepNLoop On
SL=[0 or 1]	Enable or disable StepNLoop Control	OK
SLA	Returns maximum number of StepNLoop control attempt	28-bit number
SLA=[value]	Sets maximum number of StepNLoop control attempt	OK
SLE	Returns StepNLoop correction range value.	28-bit number
SLE=[value]	Sets StepNLoop correction range value.	OK
SLR	Returns StepNLoop ratio value	[0.001 – 999.999]
SLR=[factor]	Sets StepNLoop ratio value. Must be in the range [0.001 – 999.999]	OK

SLS	Returns current status of StepNLoop control	See Table 7.8
SLT	Returns StepNLoop tolerance value	32-bit
SLT=[value]	Sets StepNLoop tolerance value.	OK
SLOAD	Returns RunOnBoot parameter	See Table 7.13
SLOAD=[0 or 1]	Set RunOnBoot parameter	See Table 7.13
SPC[0,1]	Get program counter for standalone program	[0-1784]
SR[0,1]=[Value]	Control standalone program: 0 – Stop standalone program 1 – Run standalone program 2 – Pause standalone program 3 – Continue standalone program	OK
SSPD[value]	On-the-fly speed change. In order to use this command, S-curve control must be disabled. Use SCV command to enable and disable s-curve acceleration/ deceleration control. Note that an "=" sign is not used for this command.	OK
SSPDM	Return on-the-fly speed change mode	[0-9]
SSPDM=[value]	Set on-the-fly speed change mode	OK
STOP	Stops the motor using deceleration if in motion.	OK
STORE	Store settings to flash	OK
SYNC	Read sync output configuration: 1 – trigger when encoder EQUALS position 2 – trigger when encoder is LESS than position 3 – trigger when encoder is GREATER than position	1,2,3
SYNC=	Set sync output configuration: 1 – trigger when encoder EQUALS position 2 – trigger when encoder is LESS than position 3 – trigger when encoder is GREATER than position	OK
SYNF	Turn off sync output	OK
SYNO	Turn on sync output	OK
SYNP	Get trigger position	28 bit signed number
SYNP=	Set trigger position	28 bit signed number
SYNT	Get pulse width time (ms). Only applicable if sync output configuration is set to 1.	Milli-seconds
SYNT=	Set pulse width time (ms). Only applicable if sync output configuration is set to 1. Max 30ms	OK
T[value]	On-the-fly target change	OK
TOC	Get time-out counter (ms)	32-bit number
TOC=[value]	Set time-out counter (ms)	OK
V[0-99]	Read variables 0-99	28-bit number
V[0-99]=[value]	Set variables 0-99	OK

VER	Get firmware version	Vxxx
X[value]	Moves the motor to absolute position value using the HSPD, LSPD, and ACC values.	OK
Z+	Homes the motor in positive direction using the Z index encoder channel ONLY.	OK
Z-	Homes the motor in negative direction using the Z index encoder channel ONLY.	OK
ZH+	Homes the motor in positive direction using the home switch and then Z index encoder channel.	OK
ZH-	Homes the motor in negative direction using the home switch and then Z index encoder channel.	OK

Table 9.0

9.2. Error Codes

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

Error Code	Description
?[Command]	The ASCII command is not understood.
?ABS/INC is not in operation	T[] command is invalid because a target position move is not in operation.
?DIO Enabled	Cannot control digital output because DIO mode is enabled.
?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 controller is outputting pulses.
?SA running	Cannot enable DIO mode because stand-alone is running.
?SCV ON	Cannot perform SSPD move because s-curve is enabled.
?Speed out of range	SSPD move parameter is out of the range of the SSPDM speed window.
?State Error	A move command is issued while the controller is in error state.
?Sub not Initialized	Call to a subroutine using the **GS** command is not valid because the specified subroutine has not been defined.

Table 9.1

10. Standalone Programming 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.

10.1. Standalone Command Set

Command	R/W	Description	Example
;	-	Comment notation. Comments out any text following ; in the same line.	;This is a comment
ABORTX	W	Immediately stop all motion.	ABORTX
ABS	W	Set the move mode to absolute mode.	ABS X1000 ;move to position 1000
ACC	R/W	Set/get the acceleration setting. Unit is in milliseconds.	ACC=500 ACC=V1
DEC	R/W	Set/get the deceleration setting. Unit is in milliseconds.	DEC=300 DEC=V1
DELAY=[Value]	W	Set a delay in milliseconds. Assigned value is a 32-bit unsigned integer.	DELAY=1000 ;1 second
DI	R	Return status of digital inputs. See Table 7.1 for bitwise assignment.	IF DI=0 DO=1 ;Turn on DO1 ENDIF
DI[1-6]	R	Get individual bit status of digital inputs. Will return [0,1]. See Table 7.1 for bitwise assignment.	IF DI1=0 DO=1 ;Turn on DO1 ENDIF
DO	R/W	Set/get digital output status. See Table 7.2 for bitwise assignment.	DO=2 ;Turn on DO2
DO[1-2]	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
DRVIC	W	Sets the driver idle current parameter. Units are in mA.	DRVRC=500 ;Set idle current to 500mA
DRVIT	W	Sets the driver idle time parameter. Units are in cent-sec.	DRVIT=1 ;Idle-time set to 1 cent-sec
DRVMS	W	Sets the driver micro-step parameter.	DRVMS=100 ;Set micro-step to 100
DRVRC	W	Sets the driver run current parameter. Units are in mA.	DRVRC=1000 ;Set run current to 1000mA
ECLEARX	W	Clear any motor status errors. See Section 7.17.5 for types of errors.	ECLEARX
EO	R/W	Set/get the enable output status.	EO=1 ;Enable the motor
EX	R/W	Set/get the current encoder position.	EX=0 ; Set the encoder position to 0
GOSUB [0-31]	-	Call a subroutine that has been previously stored to flash memory.	GOSUB 0 END
HLHOMEX[+/-]	W	Home the motor using the home input at low and high speeds in the specified direction. See section 7.9.3 for details	HLHOMEX+ ;positive home WAITX ;wait for home move
HOMEX[+/-]	W	Home the motor using the home input at high speed in the specified direction. See section 7.9.1 for details.	HOMEX- ;negative home WAITX ;wait for home move
HSPD	R/W	Set/get the high speed setting. Unit is in pulses/second.	HSPD=1000 HSPD=V1
IF ELSEIF ELSE	-	Perform a standard IF/ELSEIF/ELSE conditional. Any command with read ability can be used in a conditional	IF DI1=0 DO=1 ;Turn on DO1 ELSEIF DI2=0

ENDIF		ENDIF should be used to close off an IF statement.
		
		Conditions [=, >, <, >=, <=, !=] are available
			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
JOGX[+/-]	W	Move the motor indefinitely in the specified direction.	JOGX+ JOGX-
LHOMEX[+/-]	W	Home the motor using the limit inputs in the specified directions. See section 7.9.4 for details.	LHOMEX+ ;positive home WAITX
LSPD	R/W	Set/get the low speed setting. Units are in pulses/second.	LSPD=100 LSPD=V3
LTX	W	Set the latch enable.	LTX=1 ;Enable latch
LTEX	R	Get the latch encoder value.	V3=LTEX ;Get latch encoder value
LTPX	R	Get the latch position value.	V4=LTPX ;Get latch position value
LTSX	R	Get the latch status.	V2=LTSX ;Get the latch status
MSTX	R	Get the current motor status of the motor. See Table 7.0 for motor status assignment.	V1=MSTX ;Set V1 to the MST
PRG [0-1] END	-	Used to define the beginning and end of a main program. The DMX-UMD can have up to two main programs.	PRG 0 ;main program END
PS	R	Get the current pulse speed.	V1=PS ;Sets V1 to pulse speed
PX	R/W	Set/get the current motor position.	PX=1000 ;Set to 1000 V1=PX ;Read current position
RW	W	Start driver write operation. Note that after executing RW, wait 2 seconds before any other operation is executing (using DELAY=2000).	RW
RWSTAT	R	Get the driver write operation status.	V1=RWSTAT
SCVX	W	Set the s-curve enable.	SCVX=1 ;Enable s-curve
SLX	W	Get the StepNLoop status.	SL=1 ;Enable StepNLoop
SLSX	R	Get the StepNLoop status.	V1=SLSX ;Set V1 to the SNL status
SSPDX	W	Set on-the-fly speed change for an individual axis. Range is from 1 to 6,000,000 PPS.	SSPDX=3000 ;Change speed to 3000
SSPDMX	W	Set individual on-the-fly speed change mode. Range is from 0 to 9.	SSPDMX=1 ;Set speed change mode
SR[0,1]=[Value]	W	Set the standalone control for the specified program. See Section 7.17.2.	SR0=0 ;Turn off program 0
STOPX	W	Stop all motion using a decelerated stop.	STOPX
STORE	-	Store settings to flash.	STORE
SYNCFGX	W	Set the sync output configuration.	SYNCFGX =1 ;Set sync configuration
SYNOFFX	W	Disable the sync output feature.	SYNOFFX ; Disable sync output feature
SYNONX	W	Enable the sync output feature.	SYNONX ;Turn on sync output feature
SYNPOSX	W	Set sync output position.	SYNPOSX=3000 ;Set sync position to 3000
SYNSTATX	R	Get the status for sync output.	V1=SYNSTATX; Set V1 to the sync status
SYNTIMEX	W	Set the pulse output width time for sync output.	SYNTIMEX=10 ;Set sync pulse time

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 are in milliseconds.	TOC=1000 ;1 second time-out
V[0-99]	R/W	Set/get standalone variables. The following operations are available: [+] Addition [-] Subtraction [*] Multiplication [/] Division [%] Modulus [>>] Bit shift right [<<] Bit shift left [&] Bitwise AND [|] Bitwise OR [~] Bitwise NOT	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
WAITX	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 motor to [position]. If in incremental mode, move the motor to [current position] + [position].	X1000
ZHOME[+/-]	W	Perform Z-homing using the current high speed, low speed, and acceleration. See section 7.9.2 for details.	ZHOME+ ZHOME-
ZOME[+/-]	W	Perform Zoming (homing only using Z-index) using the current high speed, low speed, and acceleration. See section 7.9.5 for details.	ZOME+ ZOME-

Table 10.0

10.2. Example Standalone Programs

10.2.1. 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
END             ;* End of the program
```

10.2.2. 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
    X1000       ;* Move to 1000
    WAITX       ;* Wait for X-axis move to complete
    X0          ;* Move to zero
ENDWHILE        ;* Go back to WHILE statement
END
```

10.2.3. 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
    X1000       ;* Move to 1000
    WAITX       ;* Wait for X-axis move to complete
    X0          ;* Move to zero
    V1=V1+1     ;* Increment variable 1
ENDWHILE        ;* Go back to WHILE statement
END
```

10.2.4. 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
        X1000           ;* Move to 1000
        WAITX           ;* Wait for X-axis move to complete
        X0              ;* Move to zero
    ENDIF
ENDWHILE          ;* Go back to WHILE statement
END
```

10.2.5. 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         ;* Jump to subroutine 1
    ENDIF
ENDWHILE                ;* Go back to WHILE statement
END

SUB 1
    XV1                 ;* Move to V1 target position
    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
```

10.2.6. 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
        ELSEIF DI2=1    ;* If digital input 2 is on
                X2000   ;* Move to 2000
        ELSEIF DI3=1    ;* If digital input 3 is on
                X3000   ;* Move to 3000
        ELSEIF DI5=1    ;* If digital input 5 is on
                HOMEX-  ;* Home the motor in negative direction
        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
```

10.2.7. 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 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
```

10.2.8. 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 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&1024    ;* 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
```

Appendix A: Speed Settings

HSPD value [PPS] †	Speed Window [SSPDM]	Min. LSPD value	Min. ACC [ms]	δ	Max ACC setting [ms]
1 - 16 K	0,1	10	2	500	((HSPD – LSPD) / δ) × 1000
16K - 30 K	2	10	1	1 K	
30K - 80 K	3	15	1	2 K	
80K - 160 K	4	25	1	4 K	
160K - 300 K	5	50	1	8 K	
300K - 800 K	6	100	1	18 K	
800K - 1.6 M	7	200	1	39 K	
1.6 M - 3.0 M	8	400	1	68 K	
3.0 M – 6.0 M	9	500	1	135 K	

Table A.0

†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

Acceleration/Deceleration 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.

Max ACC = ((HSPD – LSPD) / δ) × 1000 [ms]

Figure A.0

Examples:

a) If **HSPD** = 20,000 pps, **LSPD** = 100 pps:
   a. Min acceleration allowable: **1 ms**
   b. Max acceleration allowable:

((20,000 – 100) / 1,000) x 1,000 ms = **19900 ms** (19.9 sec)

b) If **HSPD** = 900,000 pps, **LSPD** = 1000 pps:
   a. Min acceleration allowable: **1 ms**
   b. Max acceleration allowable:
   ((900,000 – 1000) / 39,000) x 1000 ms = **23050** ms (23.05 sec)

Acceleration/Deceleration Range – Positional Move

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

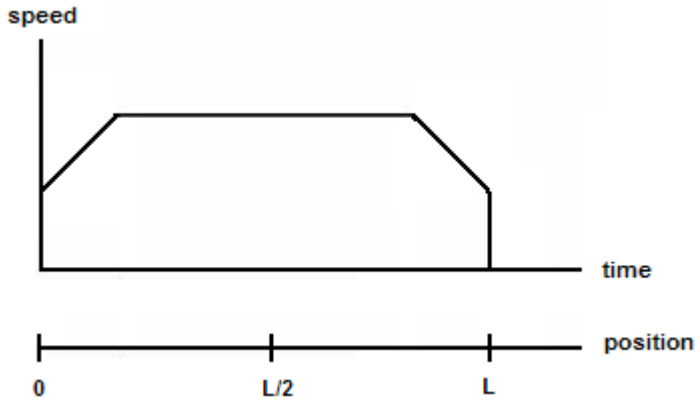


Figure A.1

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 Drive
Livermore, CA 94551
925-373-8800

www.arcus-technology.com