SPiiPlus Setup Guide

Version 6.50

Version 6.50, 30 January 2009

Web Site: **www.AcsMotionControl.com**
Information: **info@AcsMotionControl.com**
Tech Support: **support@AcsMotionControl.com**

ACS MotionControl, Ltd.
Ramat Gabriel Industrial Park
POB 5668
Migdal HaEmek, 10500
ISRAEL
Tel: (972) (4) 6546440
Fax: (972) (4) 6546443

ACS MotionControl, Inc.
6575 City West Parkway
Eden Prairie, MN 55344 USA
Tel: 800-545-2980
Tel. 763-559-7669
Fax. 763-559-0110

ACS Motion Control (Korea)
Digital Empire Building D-191
980-3, Youngtong-dong, Youngtong-gu,
Suwon, Geonggi-do, 443-813, Korea
Tel: +82-31-202-3541
Fax: +82-31-202-3542

Changes in Version 6.50

Page	Change

Table of Contents

List of Figures

List of Tables

1 Introduction

This guide describes how to configure and adjust SPiiPlus motion control products to work with supported types of motors and feedback devices.

This guide applies for the following SPiiPlus motion control product lines:

- SPiiPlus PCI-4/8 controllers
- SPiiPlus SA stand-alone controllers
- SPiiPlus CM control modules (controllers with integrated drives)

Note	
	*The term "controller" is used in this guide whenever information applies for both controllers and control modules. If information applies to only one of these product groups, the group is stated explicitly.*

Supported types of motors include:

- DC brush
- DC brushless with three phases
- Pulse-Direction stepper
- Piezo ceramic from Nanomotion (http://www.nanomotion.com/) - supported only by SPiiPlus PCI.

Supported types of feedback devices include:

- quadrature encoder
- pulse-direction encoder
- up-down encoder
- sin-cos encoder (ordering option)
- analog input
- rotary and linear Electric Encoders™ from Netzer Precision (http://www.netzerprecision.com/)

This guide describes the procedures for setting up SPiiPlus controllers as summarized below:

1.1 Organization of this Guide

- **Chapter 2 - "SPiiPlus Hardware Connection"**
- **Chapter 3 - "SPiiPlus Software Installation"**
- **Chapter 4 - "Communication Setup"**
- **Chapter 5 - "Axis Configuration and Setup"**
- **Chapter 6 - "Current Loop Adjustment"**
- **Chapter 7 - "Commutation for DC Brushless (AC Servo) Motors"**
- **Chapter 8 - "Open Loop and Index Verification"**
- **Chapter 9 - "Position and Velocity Loops"**
- **Chapter 10 - "Motion Defaults"**
- **Chapter 11 - "Saving, Copying, and Protecting Axis Variables"**
- **Chapter 12 - "Homing Programs"**
- **Chapter 13 - "Appendix A: Nanomotion Piezo Ceramic Motors"**
- **Chapter 14 - "Appendix B: Netzer Precision Electric EncoderS™"**
- **Chapter 15 - "Appendix C: Working with a Gantry Axis"**
- **Chapter 16 - "Appendix D: Working with Dynamic Braking"**
- **Chapter 17 - "Appendix E: ASDF Measuring Motion Performance"**

1.2 Related SPiiPlus Tools

Table 1 Related SPiiPlus Tools

Tool	Description
SPiiPlus MMI	A multipurpose user interface with the controller including: Program management, Motion management, Communication terminal, Four channel digital oscilloscope, Safety and I/O signals monitor, Signal tuning and adjustment, and a fully interactive simulator.
SPiiPlus SPiiDebugger	A developing and debugging environment for real-time motion control algorithms inside of SPii processor.
SPiiPlus Utilities	The SPiiPlus Upgrader allows upgrading or downgrading of the controller firmware. The SPiiPlus Emergency Wizard allows firmware recovery in case of damage or loss of communication to the controller.

1.3 The SPiiPlus Documentation

Table 2 Collateral Documentation

Document	Description
SPiiPlus PCI-4-8 Hardware Guide	Installation and hardware connection with the SPiiPlus PCI 4 or 8 axes
SPiiPlus CM Hardware Guide	Installation and hardware connection with the SPiiPlus Control Module
SPiiPlus Setup Guide	Communication, configuration and adjustment procedures for SPiiPlus motion control products.
SPiiPlus ACSPL+ Programmer's Guide	Command set and high level language for programming SPiiPlus controllers.
HSSI Expansion Modules Guide	High-Speed Synchronous Serial Interface (HSSI) for expanded I/O, distributed axes, and nonstandard devices.
SPiiPlus Library Reference	C++ and Visual Basic® libraries for host PC applications. This guide is applicable for all the SPiiPlus motion control products
SPiiPlus Utilities User's Guide	Firmware upgrade and recovery procedures.
SPiiPlus COM Library Reference Guide	COM Methods, Properties, and Events for Communication with the Controller
SPiiPlus FRF Analyzer User's Guide	The SPiiPlus FRF (Frequency Response Function) Analyzer™ is a powerful servo analysis GUI for ACS MotionControl SPiiPlus motion controllers.
SPiiPlus SA and SA-LT Hardware Guide	Installation and hardware connection with the SPiiPlus SA and SPiiPlus SA-LT Controllers
SPiiPlus 3U Hardware Guide	Installation and hardware connection with the SPiiPlus 3U Controller
MC4U Hardware Guide	Installation and hardware connection with the MC4U Control Module integrated motion controller.

1.4 Conventions Used in this Guide

Several text formats and fonts, illustrated in **Table 3**, are used in the text to convey information about the text.

Table 3 Text Conventions

Text	Description
BOLD CAPS	ACSPL+ elements (commands, functions, operators, standard variables, etc.) when mentioned in the text. Software tool menus, menu items, dialog box names and dialog box elements.
bold	Emphasis or an introduction to a key concept.
`Monospace`	Code examples.
`Italic monospace`	Information in code examples that the user provides.
ALL CAPS	(Keyboard) key names [example: SHIFT key].
Bold Blue Text	Links within this document, to web pages, and to e-mail addresses.
|	Used in command syntax to indicate input of one alternative *or* another.
→	Used in GUI descriptions to indicate nested menu items and dialog box options leading to a final action. For example, the sequence: **Debug → New Watch → Real-time** directs the user to open the Debug menu, choose the New Watch command, and select the Real-time option.

1.5 Statement Text and Icons Used in this Guide

Note 	*Notes include helpful information or tips.*

Caution 	***A Caution describes a condition that may result in damage to equipment.***

Warning 	***A Warning describes a condition that may result in serious bodily injury or death.***

Advanced 	*Indicates a topic for advanced users.*

Model 	*Highlights a specification, procedure, condition, or statement that depends on the product model.*

1.6 Required Knowledge for Using This Guide

SPiiPlus controllers are programmed using ACSPL+, a proprietary high level language optimized for motion control. Familiarity with ACSPL+ is necessary for working with the controller. ACSPL+ is described in detail in the *SPiiPlus ACSPL+ Programmer's Guide*.

ACSPL+ libraries are provided for host programming in other high level languages. The library for C, C++, and Visual Basic are described in the *SPiiPlus C Library Reference*.

2 SPiiPlus Hardware Connection

Installation and the electrical interface information for the controller is provided in the hardware guide and includes the following:

- Installation: either standalone or PC-based*
- Power supply connection*
- Communication (with a PC) connection*
- Encoder connection (for closed-loop control) – per axis*
- Direct connection to a servo drive (controller or control module) or to a servo motor (control module only) – per axis*
- Limit switch connections – per axis
- Emergency stop button connection
- Pulse-direction stepper drive connection – per axis
- Digital I/O connection
- Analog I/O connection
- HSSI-network module connections (remote axes and I/O expansion)

** required*

3 SPiiPlus Software Installation

All SPiiPlus motion control products use the same software (firmware for controller, high-level language libraries for host programs, and Windows-based tools for host-based controller setup and controller program development).

SPiiPlus software can be installed on Windows ® 95, 98, NT, 2000, ME and XP.

Each new version of SPiiPlus software is installed in a new folder (with the new version name) under **C:\Program Files\ACS MotionControl\SPiiPlus x.xx**.

The SPiiPlus ADK (Advanced Development Kit) installation CD includes:

Firmware – system-level software that runs on the controller's MPU (motion processing unit). The version of the firmware is identical to the version of the SPiiPlus CD installation and supplied for maintenance/backup purposes. Each product is provided by ACS with the most updated firmware released version (unless otherwise is required).

SPiiPlus MMI – Windows-based software tool for configuring, adjusting, programming and viewing variables.

SPiiPlus C Libraries – C, C++, and Visual Basic libraries for PC host programming in these high level languages. Includes sample projects for Visual C++ and Visual Basic.

SPiiPlus Simulator - Windows-based controller simulator that runs on the PC without any connected hardware. The **SPiiPlus Simulator** can communicate with the **SPiiPlus MMI**, **SPiiPlus MultiDebugger**, or the **SPiiPlus C Library**. It can be used to debug programs and to simulate motion trajectory (with zero following error), inputs, outputs and faults.

SPiiPlus Utilities - Windows-based software tools for upgrading or replacing firmware.

SPiiPlus Documentation – Complete documentation for the SPiiPlus software tools, the SPiiPlus C Library, and the SPiiPlus controller hardware. All the documents are in PDF format. In addition, the software tools include online help.

Training Files - Microsoft PowerPoint® presentations and ACSPL+ code examples.

MATLAB Simulink Files – Models of the SPiiPlus single and dual loop control algorithms.

Note	
	*ACS MotionControl recommends using the most updated SPiiPlus tools and documentation.*

Caution	
	***The SPiiPlus MMI, MultiDebugger and the firmware version should all come from same SPiiPlus ADK CD installation version.***

3.1 Installation

1. Insert the **SPiiPlus ADK** installation CD into the PC.
2. Follow on-screen installation instructions.
3. If an earlier version of SPiiPlus ADK is already installed, it is recommended to select a **Repair** installation.

Note	
	*First time installation includes **LabView**® runtime version. Rebooting the PC is required to complete the SPiiPlus software installation.*

3.2 Using Programming Resources

The SPiiPlus program group is accessed from the Windows **Start** menu and is used to start:

- SPiiPlus MMI
- SPiiPlus Utilities
- SPiiPlus documentation (PDF format)

The SPiiPlus installation folder contains two sub-folders of particular interest to users:

- ACSC: contains SPiiPlus C libraries for C, C++, and Visual Basic. Also includes sample projects for Visual C++ and Visual Basic.
- SPiiPlus Training: contains presentations about ACSPL+ and SPiiPlus C library programming and sample programs.

Note	
	*For the simulator to be accessible to a host application, a copy of the simulator executable file, **sb4.exe**, must be in the same folder as the application executable file.*

4 Communication Setup

The chapter covers:

- Communicating with the **SPiiPlus Simulator**
- Communicating with the physical motion control product via PCI Bus (SPiiPlus PCI only), RS-232 serial connection, or Ethernet (option). The Ethernet connection can be either point-to-point (PC direct to controller) or network (PC via LAN or other network to controller).

The supported connection types vary according to the motion control product model and options ordered.

Communication setup for each type of channel is described here using the **Communication** dialog box of the **SPiiPlus MMI** software tool. The **Communication** dialog box configures the controller's communication settings.

Once you have made the required hardware connection (except in the case of the Simulator) and configured the channel as described below, click **Connect** to establish communication.

If communication is successful, the following occur:

- In the **Communication** dialog box, the communication indicator turns green.
- SPiiPlus PCI-4/8 only: Green LED on controller board will flicker during communication.
- In the **SPiiPlus MMI Main Panel,** the **Firmware** field displays the controller firmware version and the active communication channel.
- The **Stop Programs and Motors** window opens. The window remains open for as long as **SPiiPlus MMI** is open.

Warning 	***The* Stop Programs and Motors *button is NOT an Emergency Stop. When you click on the button, it issues commands to stop motors and interrupt program execution. Auto routines continue to run.*** ***The* Stop Programs and Motors *button has no effect on the Emergency Stop inputs.***

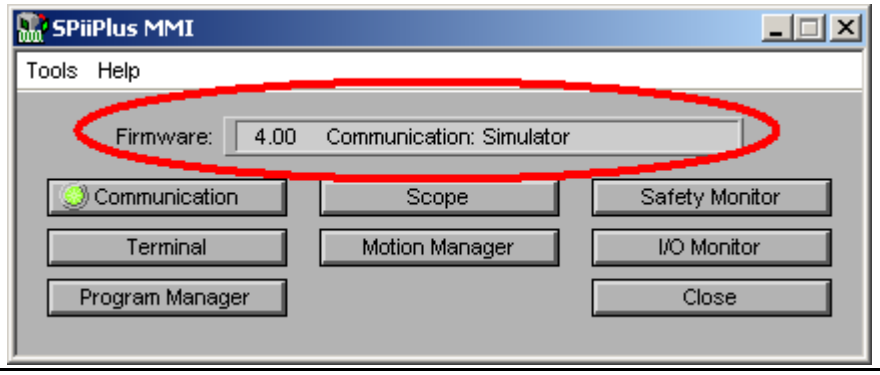


Figure 1 Main Panel – Firmware Field

Figure 2 Stop Programs and Motors Window

4.1 Communication with the Simulator

The SPiiPlus Simulator allows the user to work with the SPiiPlus MMI, SPiiPlus MultiDebugger, or SPiiPlus C Library without being physically connected to a controller, drives, or motors.

The Simulator is a powerful tool for use during application development and debugging.

The Simulator emulates all programming features of the controller, **except** for the following:

- Feedback from real motors: Instead the simulator sets the feedback values equal to the reference values, which is equivalent to ideal motors with zero following error. This means the feedback position and reference position are equal (or, in terms of the associated ACSPL+ variables: **FPOS**=**RPOS**).
- Real time operation: The simulator emulates the controller time and supplies the **TIME** variable, but 1 value of **TIME** is not equal to 1 millisecond like working with a real SPiiPlus.
- Some hardware-based programming features, including: Index, MARK, PEG, and analog I/O.
- Some immediate commands: **#HWRES**, **#RESET**, **#PROTECT**, **#UNPROTECT**, **#U**, **#IR, #SI**.
- The **write/read** commands.
- The **SPiiPlus MMI** features: **Application Saver, Application Loader**.

The Simulator provides limited support for **SPiiPlus MMI Adjuster** and **Configurator**.

4.1.1 How to Communicate with the Simulator

1. Open **SPiiPlus MMI → Communication →Simulator**.

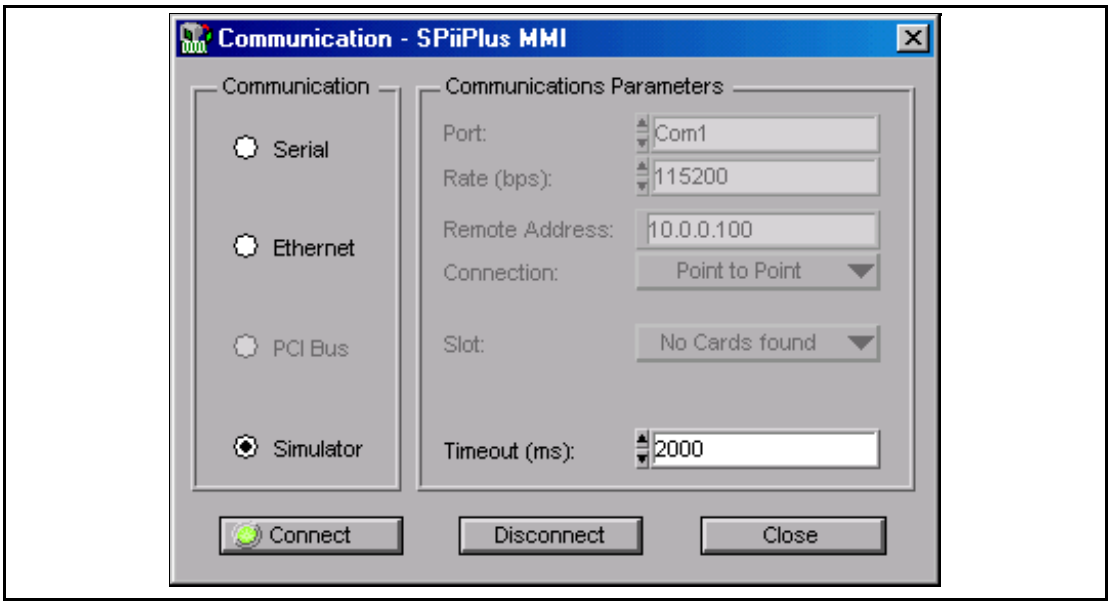


Figure 3 SPiiPlus MMI Communication Dialog - Simulator

2. Click **Connect** to establish the communication channel with the Simulator.
3. Click **Close** to exit the Communication dialog box**.**

4.1.2 Communicating with the Simulator Via Ethernet

The following procedure instructs you on how to communicate with the simulator from a host application and/or MMI via Ethernet. This form of communication allows you the following:

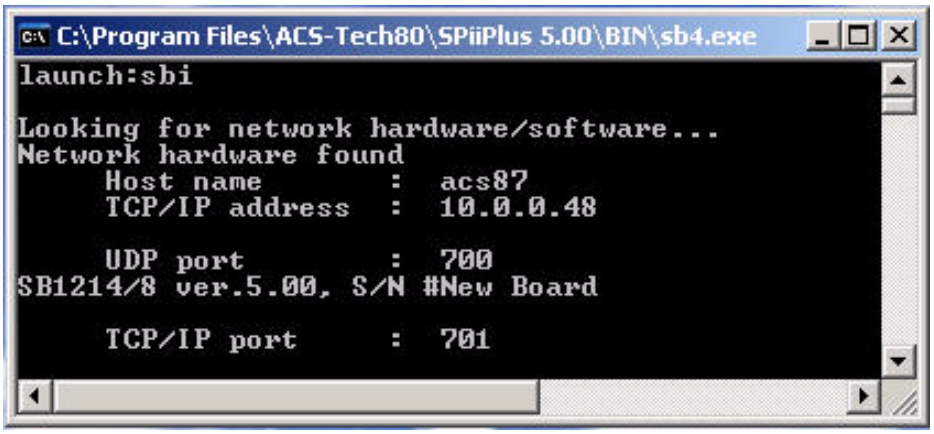
- Debugging host applications with the MMI tools. The MMI and the host application communicate with the same simulator.
- Full simulation of the controller Flash memory including Read and Write commands and using the Application Saver Loader tool.

1. Verify that the simulator (sb4.exe) resides in WINDOWS \SYSTEM or WINNT\SYSTEM32 directory.
2. On drive C:\ create the folder SB4.
3. In C:\SB4, create the following folders:
   - BAT
   - DSP
   - STARTUP
   - USER

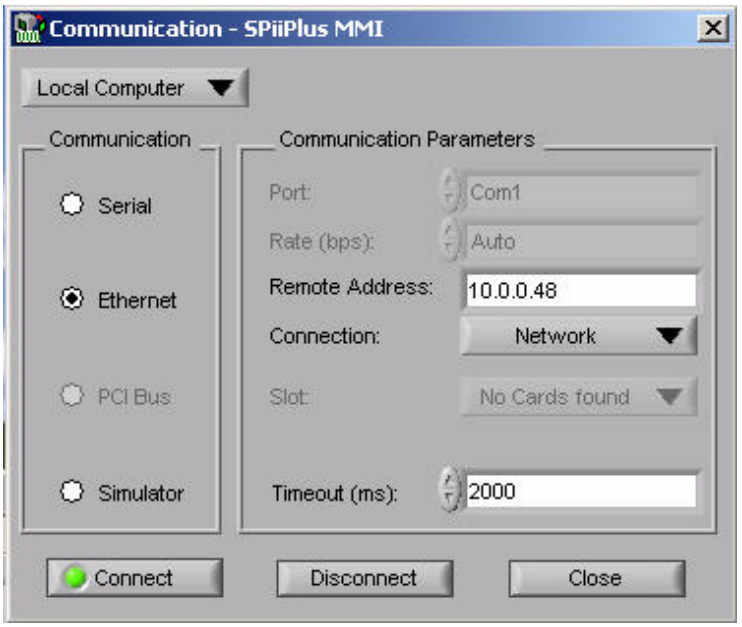
4. In C:\SB4\BAT, create the text file sbauto.bt.The text file has the following string: set network=1.
5. Execute the simulator by selecting **Start → Run** and entering the following command with the latest simulator path. For example:

   C:\Program Files\ACS MotionControl\SPiiPlus 5.00\BIN\sb4.exe" f c:\sb4 tcpd

   The simulator finds the network settings of the host and displays them on the console.



6. Connect to simulator by either of the following options:
   - From the MMI - type in the TCP/IP address displayed on the console in the Communication Dialog Remote Address field.



   - From the host application - choose and Ethernet connection with TCP/IP address displayed on the console.

4.2 Communication via PCI Bus

Model	
	*This section is only applicable for SPiiPlus PCI controllers.*

4.2.1 Configuring the Host

1. Once the SPiiPlus PCI card has been installed in a PCI slot, upon power up of the PC a **New Hardware Found** dialog box will be displayed.
2. Use the **Browse** dialog box to locate file **sb1218pci.inf**, in your most recent SPiiPlus installation directory (default path: C:\Program Files\ACS MotionControl\ SPiiPlus X.XX).

4.2.2 Establishing PCI Bus Communication

1. Open **SPiiPlus MMI → Communication → PCI Bus**. The **MMI** scans the PCI bus for installed controller cards. If at least one SPiiPlus PCI card is found, the **Slot** field becomes enabled.

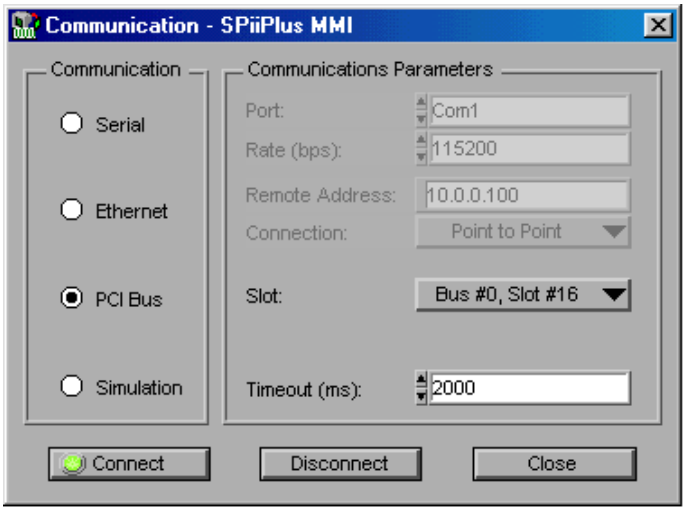


Figure 4 SPiiPlus MMI Communication Dialog - PCI

2. If multiple controller cards are installed, choose the required card in the **Slot** field.
3. Click **Connect** to establish the communication channel with the SPiiPlus PCI card.
4. Click **Close** to exit the Communication dialog box**.**

4.3 Serial RS-232/422 Communication

All SPiiPlus motion control products include two serial communication channels (COM):

- RS-232
- RS-422

The default settings of RS-232 and RS-422 are: 8 data bits, no parity and a regurlar stop bit.

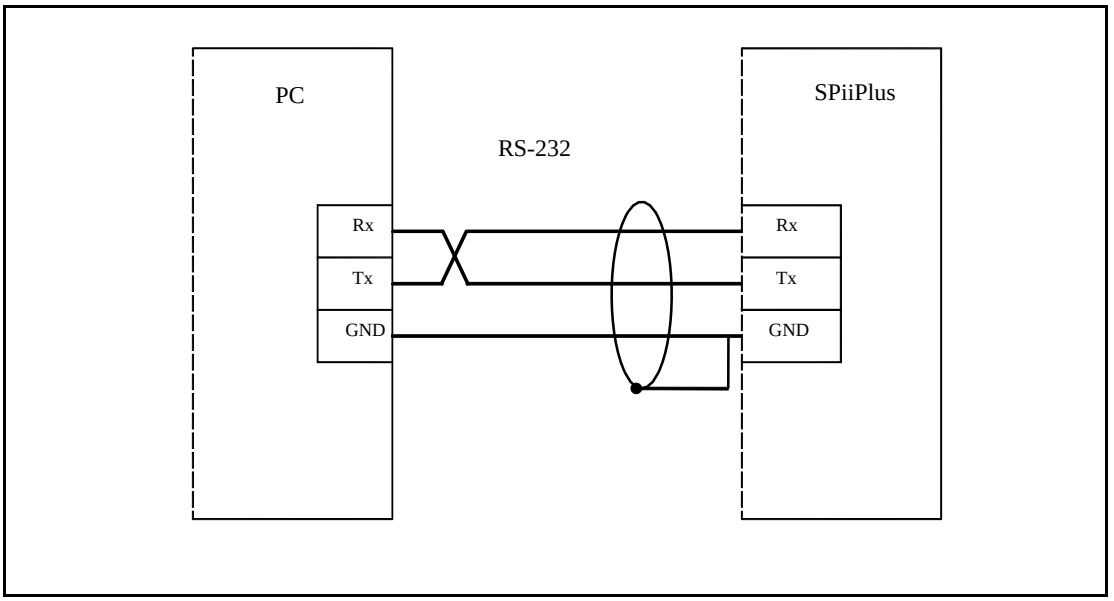


Figure 5 RS-232 Serial Connection

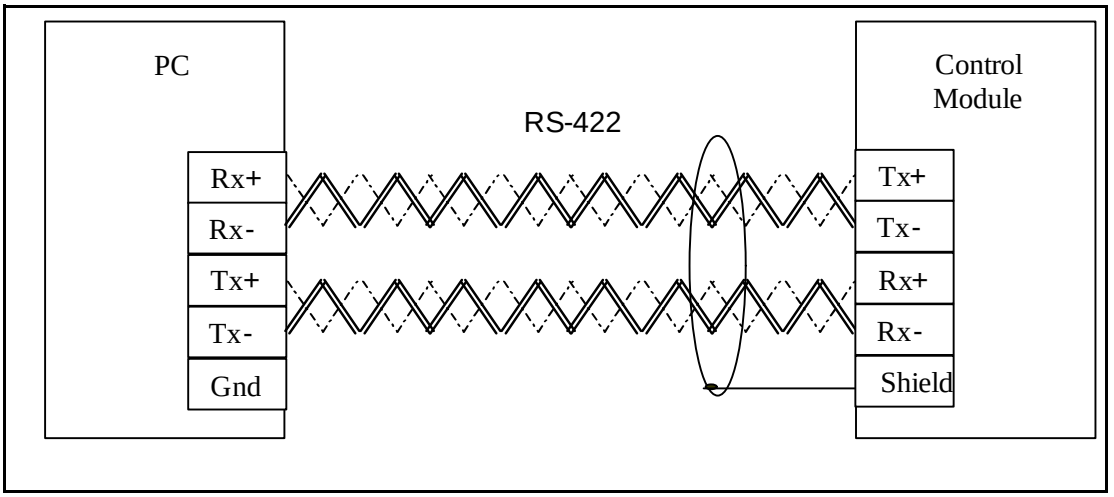


Figure 6 RS-422 Serial Connection

1. **Open MMI → Communication → Serial**.

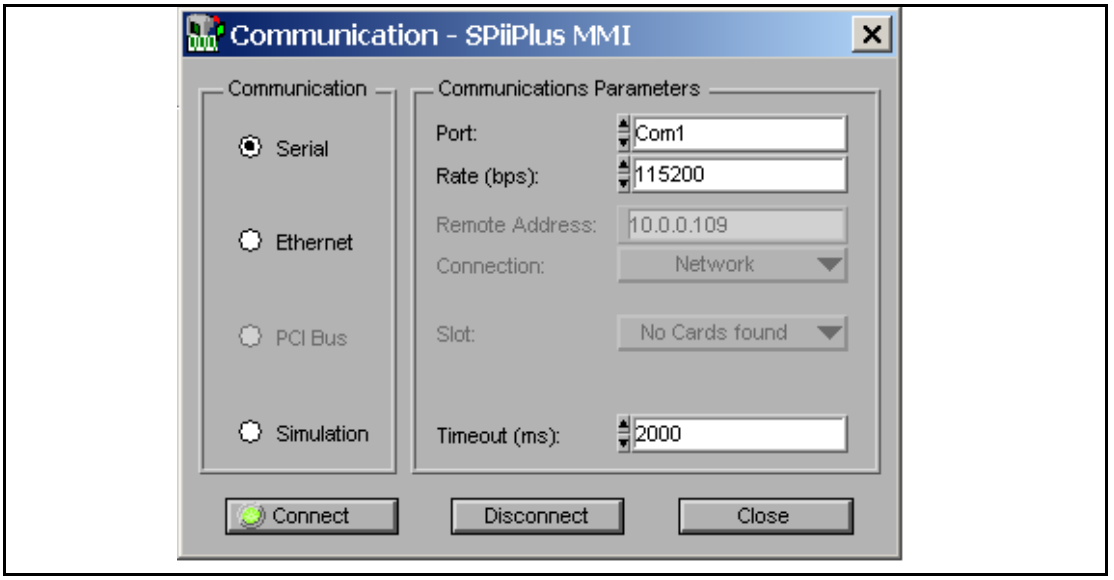


Figure 7 SPiiPlus MMI Communication Dialog - Serial

2. Specify the host computer serial **Port** (**COM1**, **COM2**, etc.) that is connected to the controller.
3. The **Rate** of the host port, which is set here, must match that of the controller port to which it is connected. Normally, this can be done by selecting **Auto**, which automatically detects the controller rate. (The controller rate can be accessed in **MMI → Configurator → Communication Parameters**.)
4. Click **Connect** to establish the communication channel with the SPiiPlus serial port.
5. Click **Close** to exit the **Communication** dialog box**.**

4.3.1 Troubleshooting a Serial Connection

- Inspect the cable and connectors.
- If a communication error message appears right after you click **Connect**, check that the COM port on the PC host is not being used by another application.
- Check that the communication port specified in the **Port** field corresponds to the COM port on the PC host that the cable is connected to.
- Older computers: try a lower baud rate.

4.4 Ethernet Communication

Ethernet is a communication option that must be ordered with the controller.

Two types of supported Ethernet connections are supported, illustrated in **Figure 8**.

1. **Network Ethernet Communication** (**Page 17**) in which the PC communicates via a network with the Ethernet port of the controller.
2. **Ethernet Point-to-Point Communication** (**Page 20**) in which the PC is connected directly via Ethernet cable to the Ethernet port of the controller

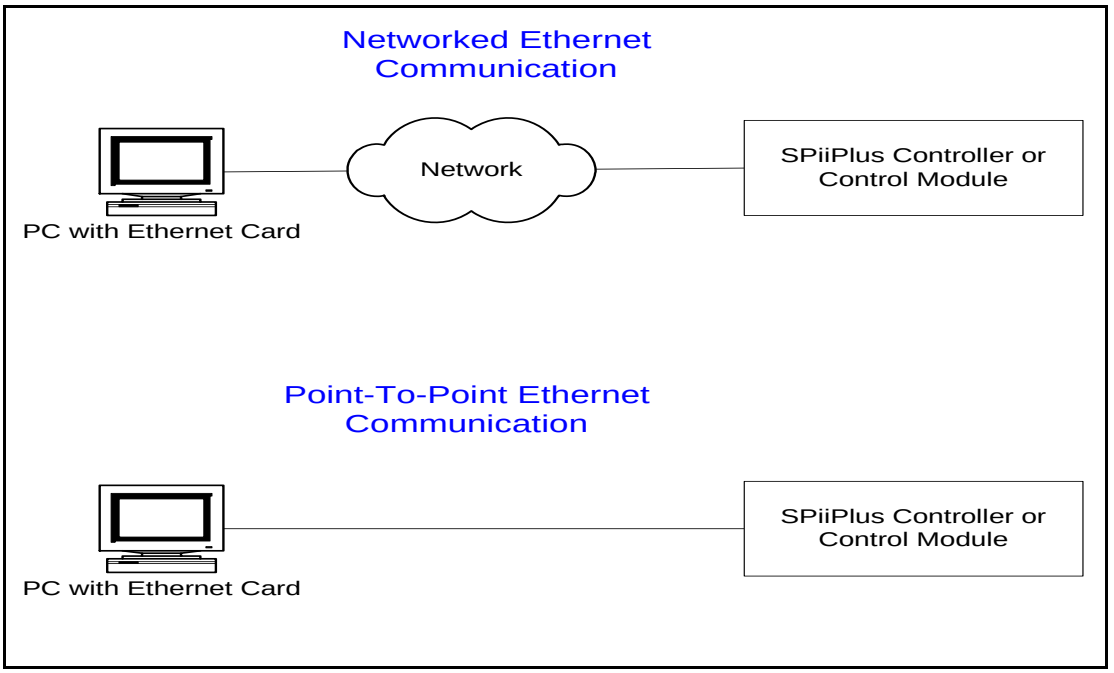


Figure 8 Types of Ethernet Connection

4.5 Network Ethernet Communication

The controller may be equipped with a 10M or 100M Ethernet adapter for a 10/100BASE-T network interface. The Ethernet interface includes one RJ-45 type connector.

If your PC host is already connected to a local Ethernet Network, you only need configure the controller and to connect it to the same network.

Note	
	*Ethernet communication is an ordering option and is not available for all products. While the physical Ethernet port may be present on the controller, it will not work unless the controller was ordered with the correct firmware!*

4.5.1 Verifying TCP/IP Support on the Host

Verify that the TCP/IP protocol is installed on your PC:

1. From the **Start** button in the Windows Task Bar choose **Settings → Control Panel → Network**. The **Network** dialog box appears.

   There will be a list of the network protocols that are installed on your PC. Depending on your version of Windows, the list may be called: “network components” (95/98) “network protocols” (NT), “connection uses the following items” (XP), etc.
2. Verify that **TCP/IP Protocol** is in the list. If it isn't, ask your network administrator to install it.
3. Verify that the **TCP/IP Protocol** is selected and click **OK**.

4.5.2 Changing the Controller TCP/IP Address

The controller cannot automatically obtain a TCP/IP address from the network server. The static TCP/IP address must be manually entered to the controller.

1. Ask the network administrator to reserve a static TCP/IP address on the network.
2. Establish serial communication with the controller (see **Section 4.3**, **Page 14**). (If the controller is a PCI model, you can establish PCI communication (see **Section 4.2**, **Page 13**) with the controller instead.
3. Open **MMI → Setup → Configurator → Communication Parameters**.

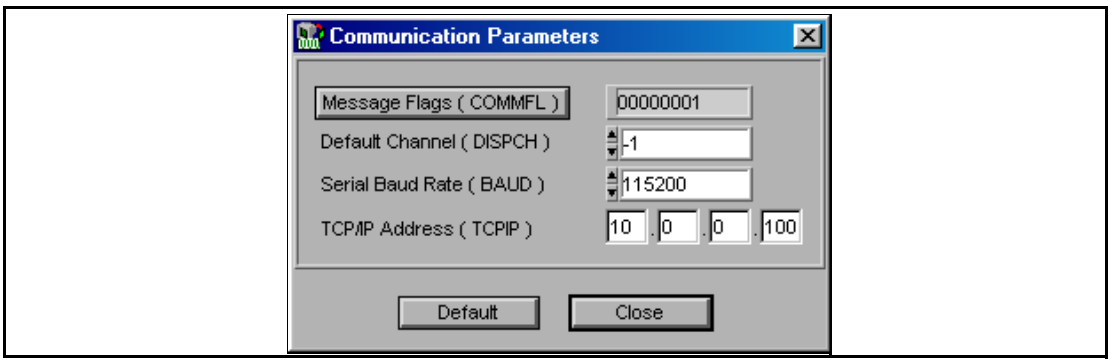


Figure 9 Communication Parameters

4. In the **TCP/IP Address** field, enter the address that you received from the network administrator (factory default is 10.0.0.100).
5. **Close** both the **Communication Parameters** and **Configurator.** The following confirmation request will appear.

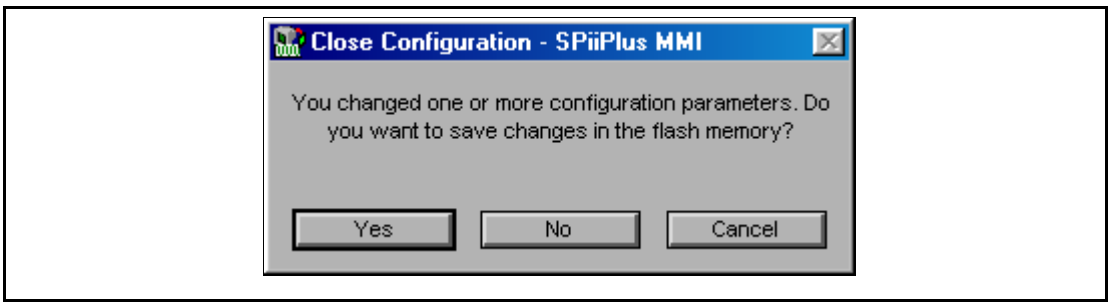


Figure 10

6. Click **Yes**. The **SPiiPlus MMI** saves the new TCP/IP address in the controller non-volatile (flash) memory.
7. The new TCP/IP address is not active immediately. To make the new value active, you must restart the controller by clicking **Application → Restart** (**#HWRES**) on the MMI Main Panel.
8. Proceed to establish Ethernet network communication.

4.5.3 Establishing Ethernet Network Communication with the Controller

1. Open **MMI → Communication → Ethernet**.

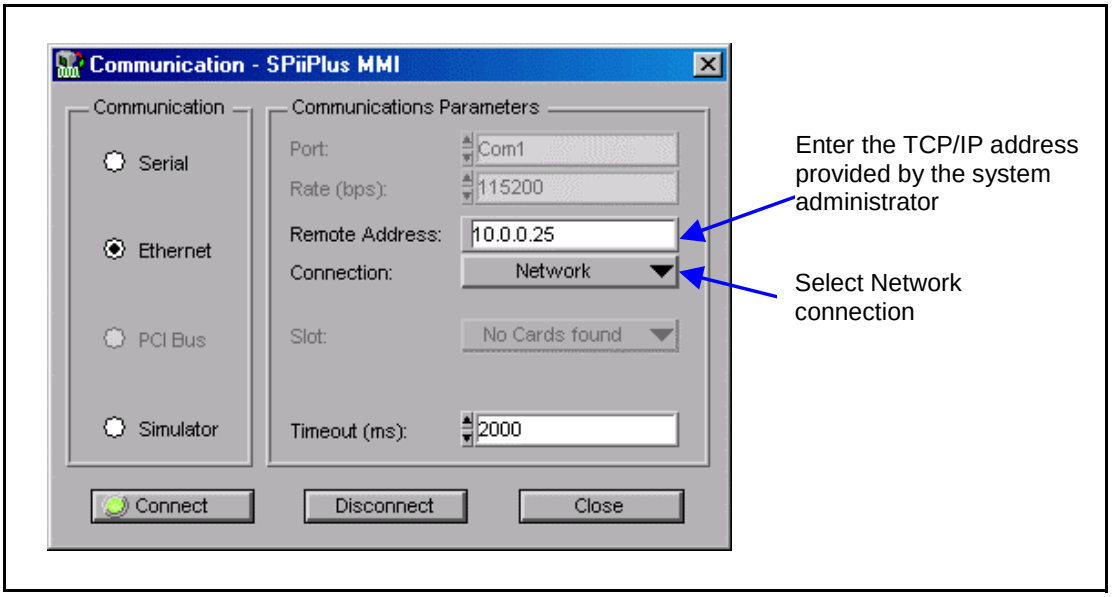


Figure 11 Communication

2. Enter the **Remote Address**, which is the TCP/IP address that you configured in the controller (factory default: 10.0.0.100).
3. In the **Connection** field select **Network**.
4. Click **Connect** to establish the network communication channel with the SPiiPlus Ethernet port.
5. Click **Close** to exit the Communication dialog box**.**

4.5.4 Troubleshooting Ethernet Network Connections

- Check that you are using Path-Through 10/100BASE-T cable.
- Inspect the cable and the connectors.
- Check that the network hardware/software (adapter, drive, TCP/IP protocol) is properly installed and configured to the PC host.
- Check that the TCP/IP address reserved for the controller is unique in the network.
- Check that the right TCP/IP address is configured in the controller.
- Check that the right TCP/IP address is entered in the **Communication** dialog box.
- If you can communicate with the controller, but an occasional error occurs, check that **Network Connection** is selected (not Point-to-Point) in the Communication dialog box.

Note	
	*If two controllers have the same IP, an error message (code 130) appears.Later, to connect via Ethernet network to the controller with a unique IP, you should restart the controller (***HWRESET***).*

4.6 Ethernet Point-to-Point Communication

If you want to make a point-to-point (direct) Ethernet connection between the PC and the controller, bear in mind that the connection requires a dedicated Ethernet card.

Note	
	*Ethernet Point-to-Point Communication requires a 10/100BASE-T Crossover cable.*

4.6.1 Configuring the Host

1. From the Start button in the Windows Task Bar choose **Settings → Control → Network**.
2. Select the TCP/IP Protocol and click OK.
3. If the TCP/IP protocol is not installed, ask your network administrator to properly install the TCP/IP protocol. If it appears several times, select the instance related to the Ethernet adaptor card that will be used for connection with the controller.
4. Open the TCP/IP Properties dialog box.

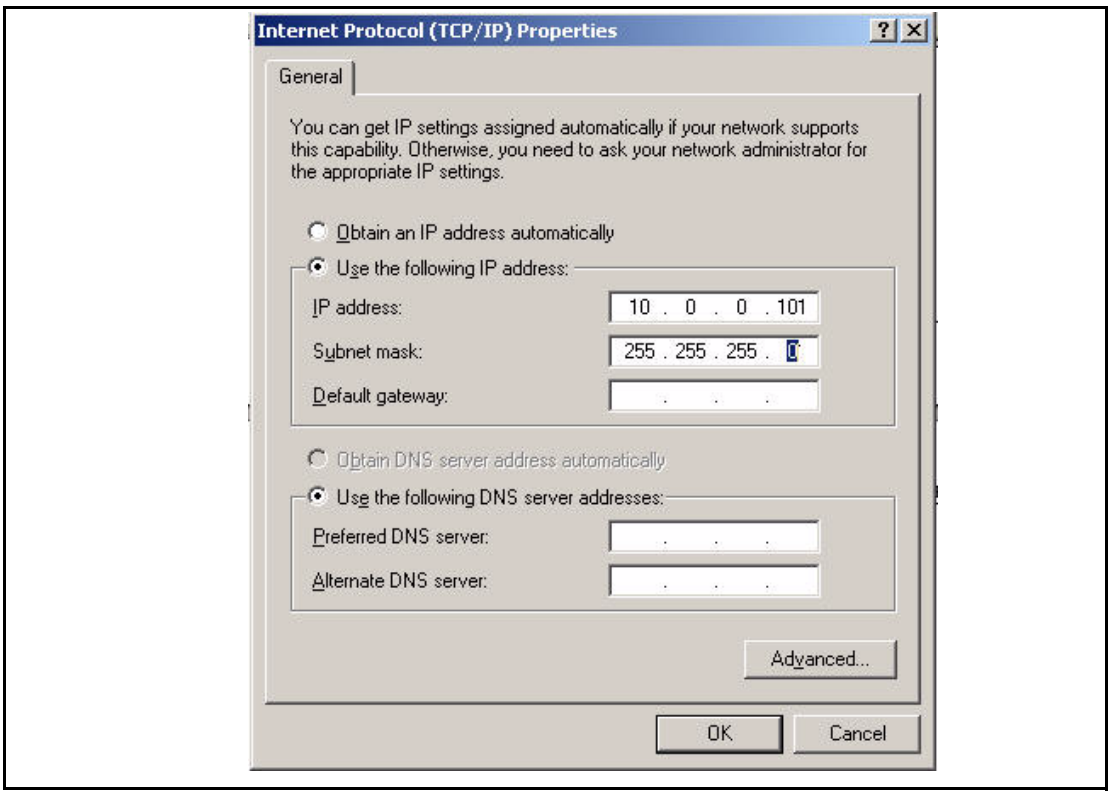


Figure 12 TCP/IP Properties Dialog

5. On the IP Address tab, select Specify an IP Address.
6. Enter the number 10.0.0.101 in the IP Address field, and 255.255.255.0 in the Subnet Mask field.

Note	*The IP Address for the PC has to be different than that of the controller.*

7. Set the other tabs as follows (recommended):

- WINS Configuration: Disable WINS Resolution
- Gateway: Clear all Installed Gateways
- DNS Configuration: Disable DNS

8. If the following dialog box appears, click Yes.

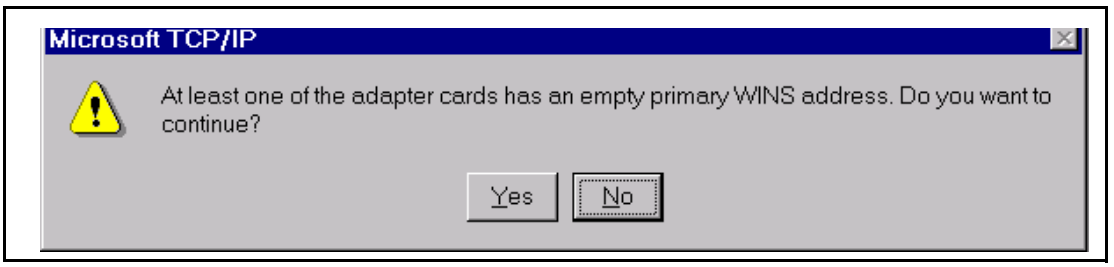


Figure 13 Microsoft TCP/IP Message

9. Restart the PC host to make the settings active.

4.6.2 Establishing Ethernet Point-to-Point Communication

1. Open **MMI → Communication → Ethernet**.

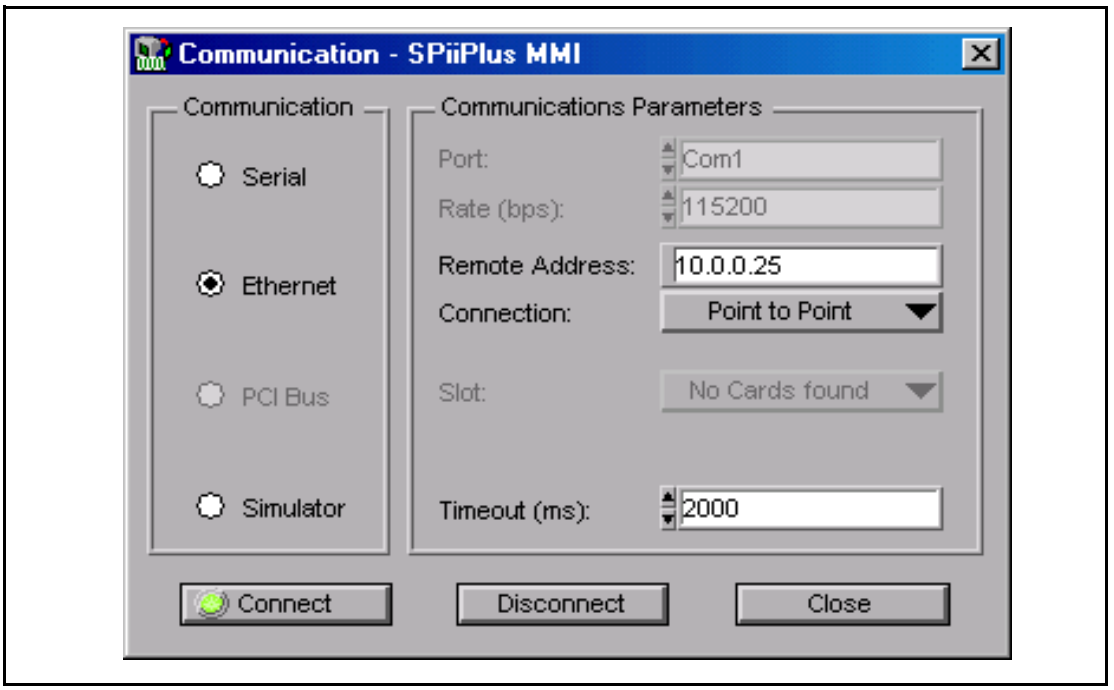


Figure 14 Communication Dialog

2. Enter the Remote Address. You must enter the TCP/IP address configured in the controller (factory default is 10.0.0.100).
3. In the Connection Parameters field select **Point-to-Point**.
4. Click **Connect**. Once communication is established the Communication indicator turns green, the Firmware box on the Main Panel displays the controller version and the communication channel being used, and the Stop Programs and Motors button appears.
5. Click **Close**.

4.6.3 Configuring the Controller

The controller is manufactured with a default TCP/IP address 10.0.0.100. You do not need to change this address in order to establish Ethernet point-to-point connection.

4.6.4 Troubleshooting Ethernet Point-to-Point Connection

- Check that you are using a Crossover 10/100BASE-T cable.
- Inspect the cable and the connectors.
- Check that the network hardware/software (adapter, drive, TCP/IP protocol) is properly installed and configured on the PC host.
- Check that the default TCP/IP address was not changed in the controller and that the proper TCP/IP address 10.0.0.100 is selected in the MMI.

5 Axis Configuration and Setup

For each axis, the controller needs to be configured and adjusted. Controller operating variables are stored in data structures called standard variables.

5.1 Selecting Axis Setup Initial Values

1. Open **MMI → Setup → Adjuster → Initial Values**.

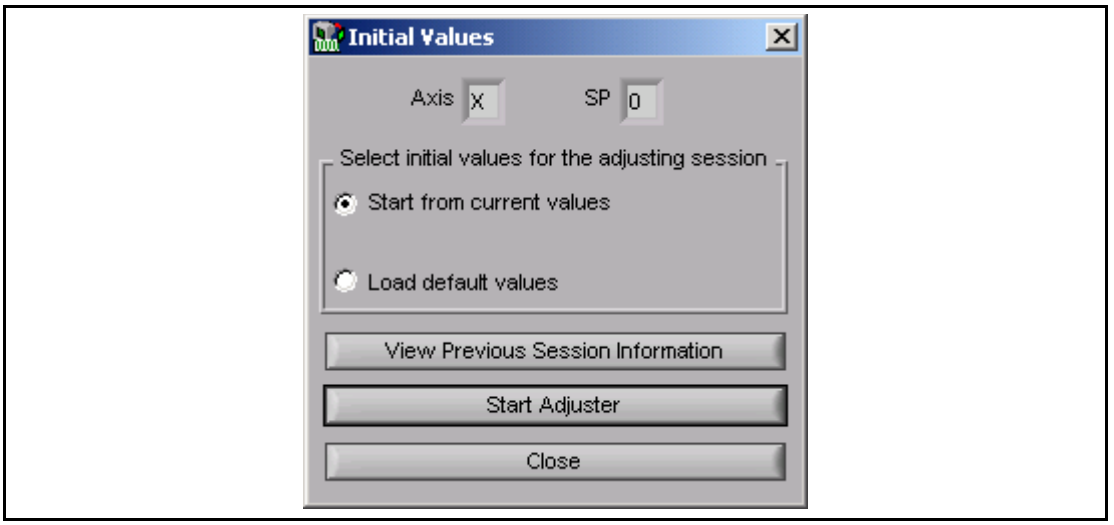


Figure 15 Initial Value Dialog

2. For an axis that has already been adjusted and requires only modification, select **Start from current values**. (To view what was changed during the last adjustment session, click **View Previous Session Information**.)

Note	
	*For an axis that has not been adjusted or where it is preferable to start from scratch, select* **Load default values**.

5.2 User Units

The units for motion travel in which the controller will be configured are user-selectable. The user unit can be millimeters, microns, nanometers, degrees or any other unit that defines a distance for a linear axis or an angle for a rotary axis.

The user unit is defined per axis. By default, the user unit is the encoder count. For example, for a quadrature encoder with resolution of 500 lines per millimeter, each default user unit equals 2µm/4 = 0.5µm.

Examples of where user units can be useful:

- Programming in standard units like millimeters or degrees is more intuitive than programming in feedback resolution units (counts).
- In a multi-axis environment, each axis could have a different feedback resolution. A motion command applied to a group of axes (for example, `PTP XYZ, 1000`) will initiate a different length of travel for each of the axes, depending on the feedback resolution. User units for each axis can be defined to compensate for different feedback resolutions, such that the command will generate the same length of travel for each axis.
- User units make it easier to change feedback devices. The program does not have to be modified for the resolution of the new feedback device.

The standard variable **EFAC** defines the user units as a function of the encoder counts.

Note	*The user units can be defined at any time in the configuration and adjustment process. It is generally recommended to define them at the beginning. This way setting values for configuration and protection variables will be easier and less prone to errors.*

Caution	***The* EFAC *variable relates to many other configuration variables. While changing the* EFAC *variable via the* MMI → Configurator*, all relevant configuration variables are being updated automatically. Therefore, we recommend changing the* EFAC *variable only via the* Configurator *and not from other places, such as* Terminal *or any application program.***

5.2.1 User Units of Measure (EFAC Variable)

1. Open **MMI → Setup → Configurator**
2. Select the axis.
3. Select the user units for the axis by defining **EFAC**. The formula to use for calculating **EFAC** depends on whether the axis is rotary or linear (see below).

5.2.1.1 Defining User Units for a Rotary Axis

$$EFAC = \frac{\text{Required user unit per mm}}{\text{Feedback resolution] lines per mm [x internal multiplier}}$$

Note	
	*For sin-cos encoders (note: support for these is an ordering option), the internal multiplier is user-configured. The controller's internal encoder multiplier is derived from the* **E_SCMUL** *variable, which defines the multiplier as a power of 2 (***MMI → Setup → Configurator → Encoder Parameters***).* *For A quad B encoders, the internal multiplier is fixed at 4.*

5.2.1.2 Defining User Units for a Linear Axis

$$EFAC = \frac{\text{Required user unit per mm}}{\text{Feedback resolution [lines per mm] x internal multiplier}}$$

5.2.1.3 Examples: User Unit Definition

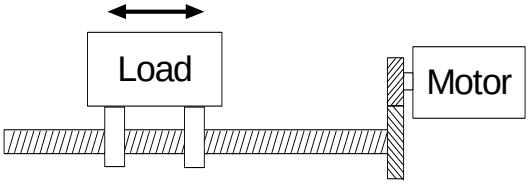
- Example 1 (rotary motor):
  - Feedback resolution: 2000 lines/rotation
  - Internal multiplier: x4
  - Required user unit: 360 degrees per rotation

  EFAC = 360/(2000x4) = 0.045
- Example 2 (linear motor):
  - Feedback resolution: 2 micron (500 lines/mm)
  - Internal multiplier: x4
  - Required user unit: mm

  EFAC = 1/(500x4) = 0.005
- Example 3 (linear motor, sin-cos encoder):
  - Encoder resolution: 500 lines/mm
  - Internal multiplier: x64
  - Required user unit: mm

  EFAC = 1/(500x64) = 0.00003125

- Example 4 (rotary motor connected to ball screw):



- Feedback resolution: 2000 lines/ rotation
- Gear ratio: 1 motor rotation=500µm (0.5mm) motion of load
- Internal multiplier: x4
- Required user unit: µm (micron)

EFAC = 1/(2000 x4 x(1/500))= 0.0625

5.2.2 Updating Other Variables with a Changed EFAC Value

Some other standard variables depend on the EFAC value. To automatically scale these variables after changing EFAC, click **Yes** when the following message appears:

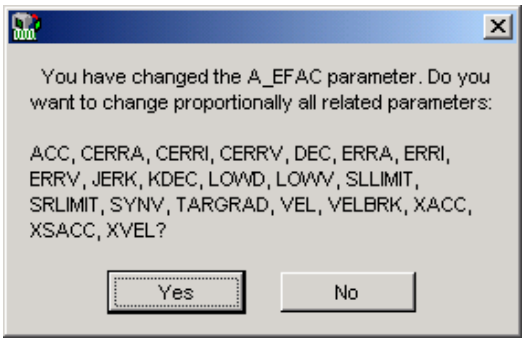


Figure 16 EFAC Parameter Message

5.2.3 Verifying User Units

Note	
	1. *These procedures are applicable only if the motor can be moved by hand.* 2. *If the feedback counting direction is the reverse of what you expect, (for example the count is positive when the motor rotates counterclockwise whereas you expect a negative count), the count can be inverted – see* **Chapter 8 - "Open Loop and Index Verification"** .

5.2.3.1 For a Rotary Axis

1. Open **MMI → Motion Manager**. Select the axis and click **Zero Feedback Position**.
2. Rotate the motor one full revolution by hand.
3. Check the **Feedback Position**. It should match the defined user unit. For example, if the user unit is degrees, then the feedback position should be around 360 (or -360).

5.2.3.2 For a Linear Axis

1. Open **MMI → Motion Manager**. Select the axis and click **Zero Feedback Position**.
2. Measure a known distance (for example, 10cm) from the axis current position.
3. Mark the axis current position and the target position.
4. Move the axis by hand to the target position.
5. Check the **Feedback Position**. The feedback position should match the defined user unit. For example, if the measured distance was 10cm and the user unit is one micron, the feedback position should be around 1,000,000 (or –1,000,000).

5.3 Protections and Faults

For a detailed description of safety-related features of SPiiPlus controllers, see Safety Control in the SPiiPlus ACSPL+ Programmer's Guide.

5.3.1 Emergency Stop Logic

Once the motion control application hardware has been connected to the controller, you must verify that the controller properly identifies the emergency stop signal.

Warning 	***You must verify that the emergency stop circuit and the controller's emergency stop logic are working correctly before initiating any motor movement!***

For information about the emergency stop hardware connection, refer to the controller's hardware guide.

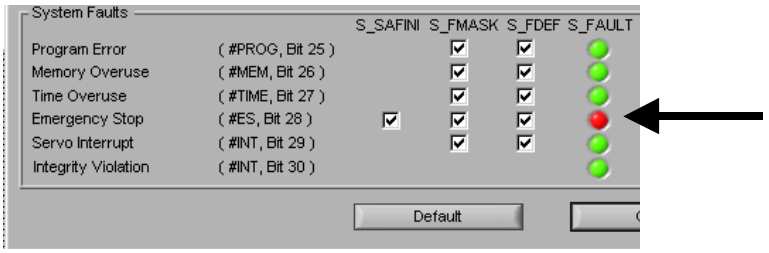
5.3.1.1 Verifying Emergency Stop Logic

1. Power on the controller.
2. Open the **SPiiPlus MMI** and establish communication (see **Chapter 4 - "Communication Setup"**) with to the controller via any communication channel.
3. From the **Main Window** click **Safety Monitor**.

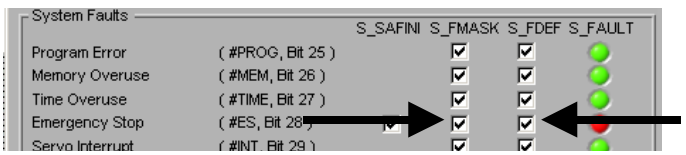
4. In the **Safety Monitor** window, locate the **Emergency Stop Input** indicator.

5. Release the physical emergency stop switch.
6. In the **Safety Monitor** window, observe the effect on the color of the **Emergency Stop Input** indicator (red or gray depending on the logic).
7. Activate the physical emergency stop switch.
8. In the **Safety Monitor** window, verify that the color of the **Emergency Stop Input** indicator has changed state as a result (from red or gray or vice versa, depending on the logic).

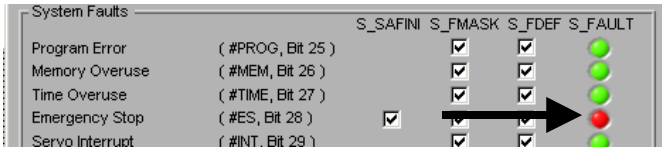
9. From the **Main** window, open the **Adjuster** from the **Setup menu**.
10. Select an axis and click **Safety Parameters**.
11. In the **Safety Parameters** window, in the **System Faults** group, locate the **Emergency Stop** line.



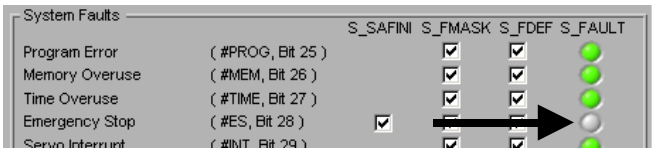
12. In the **Emergency Stop** line, select the associated **S_FMASK** and **S_FDEF** bits. (The **S_FMASK** bit determines that the controller will check periodically for the emergency stop fault, the **S_FDEF** bit determines that if the emergency stop fault is detected, the default handling process will automatically be applied.)



13. Release the physical emergency stop switch.
14. In the **Safety Parameters** window, in the **Emergency Stop** line, observe the associated **S_FAULT** indicator. (**S_FAULT** reports the existence of the associated fault).



15. In the **Safety Parameters** window, if the **S_FAULT** indicator for **Emergency Stop** is red, select the associated **S_SAFINI** bit and verify that as a result, the **S_FAULT** indicator is now gray. (**S_SAFINI** inverts the logic of the system fault inputs. For example, if a fault signal is wired so that the high signal state indicates the fault, when the corresponding **S_SAFINI** bit is set, the low signal state will indicate the fault.)



16. Activate the physical emergency stop switch.
17. Verify that the **S_FAULT** indicator turned red as a result.
18. Release the physical emergency stop switch.
19. Verify that the **S_FAULT** indicator turned green as a result.

5.3.2 Axis Limit Logic

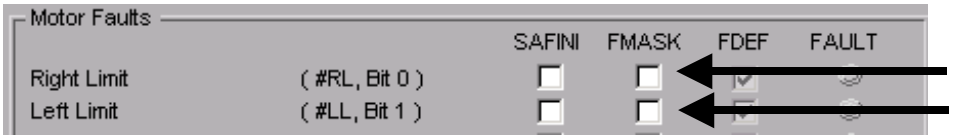
For information about the limit switch hardware connections, refer to the controller's hardware guide.

The limit procedures described here apply for both left and right limits and for each axis.

Warning	
	***If your application includes limit switches, you must verify that the limit switch circuits and the controller's limit switch logic are working correctly before initiating any motor movement.***

If your application does not include physical limit switches, proceed as follows:

1. Open **MMI → Setup → Adjuster**.
2. Select the axis and click **Safety Parameters**.
3. In the Motor Faults group, clear the **FMASK** bits for the **Right Limit** and **Left Limit**. (The default value for these bits is 1, which would cause the controller to poll the limit switch inputs for faults generated by nonexistent switches.)



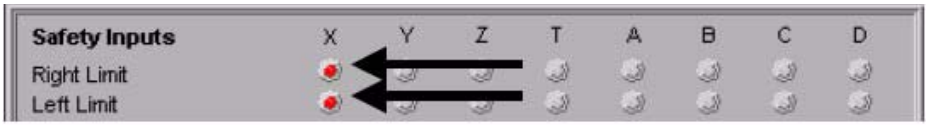
If your application does include physical limit switches, you must verify that the controller properly identifies the left (counting down) and right (counting up) limit switch signals. The following sections describe how to do so.

5.3.2.1 Verifying Axis Limit Switch Logic

Note	
	*The names of variables used with axis level variables, such as limits, are the same as those used with system level variables, such as emergency stop. The difference is that the system variables begin with "S_". For example,* **FMASK** *is the axis-level variable related to* **S_FMASK***!*

1. Open **MMI → Safety Monitor**.
2. Verify that the physical limit switch is released.
3. In the **Safety Monitor** window, note the color of the (**Right** or **Left**) **Limit** indicator (red or gray).

4. Activate the physical limit switch.
5. In the **Safety Monitor** window, verify that the color of the **Limit** indicator has changed state as a result (from red or gray or vice versa, depending on the logic).



6. Open **MMI → Setup → Adjuster**.
7. Select any axis and click **Safety Parameters**.
8. In the **Right Limit** (or **Left Limit**) line select the associated **FMASK** and **FDEF** bits. (The **FMASK** bit determines that the controller will check periodically for the limit fault, the **FDEF** bit determines that if the limit fault is detected, the default handling process will automatically be applied.)

9. Release the physical limit switch.
10. In the **Safety Parameters** window, if the **FAULT** indicator for the **Limit** is red, select the associated **SAFINI** bit and verify that as a result, the **FAULT** indicator is now gray. (**SAFINI** inverts the logic of the system fault inputs. For example, if a fault signal is wired so that the high signal state indicates the fault, when the corresponding **SAFINI** bit is set, the low signal state will indicate the fault.)
11. Activate the physical limit switch.
12. Verify that the **FAULT** indicator turned red as a result.
13. Release the physical limit switch.
14. Verify that the **FAULT** indication turned gray as a result.

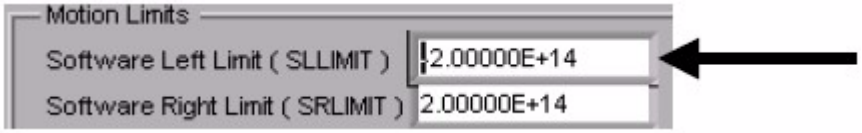
5.3.2.2 Setting Axis Left And Right Software Limits

When the motion of the axis has limited travel between two physical boundaries (hard stops), the user can define left (when counting down) and right (when counting up) software limit positions.

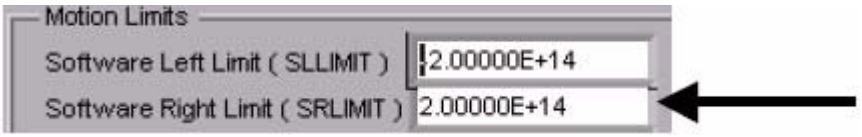
Typically a software limit would be set to serve as a warning before the axis reaches the corresponding physical limit switch. The software limits are defined in user units.

Note	*The axis is moved manually in the following procedure. If this is not possible, it is recommended waiting until the motor is configured and then to jog the motor to the desired software limit position.*

1. Open **MMI ➔ Motion Manager** and move the physical axis manually to the zero position.
2. In the **Motion Manager** window, click **Zero Feedback Position**.
3. Move the axis manually to the left (counting down) until you reach the position that you want to define as the software left limit.
4. In the **Motion Manager** window, find the **Feedback Position (FPOS)** and record it.
5. Open **MMI ➔ Setup ➔ Adjuster**.
6. Select the axis and click **Safety Parameters**.
7. In the **Safety Parameters** window, in the **Software Left Limit** field, enter the position that you recorded in step.



8. Move the axis manually to the right (counting up) until you reach the position that you want to define as the software right limit.
9. In the **Motion Manager** window, again find and record the **Feedback Position (FPOS)**.
10. In the **Safety Parameters** window, in the **Software Right Limit** field, enter the new feedback position that you recorded.



5.3.3 Axis Following Errors

A following error occurs when there is a difference between the reference (desired) position (**RPOS**) and the feedback (actual) position (**FPOS**). A following error fault occurs when the following error exceeds a user-defined threshold. Following errors are defined per axis and in user units.

There are two following error faults:

- **PE** (position error): A following error fault that is still considered part of normal operation. Normally, the user can program a condition based on this fault. See more in the *SPiiPlus ACSPL+ Programmer's Guide*.
- **CPE** (critical position error): A following error fault that is considered an abnormal operation. See more in the *SPiiPlus ACSPL+ Programmer's Guide*.

Each of these faults has three user-defined thresholds. The applicable threshold depends on the current motor state:

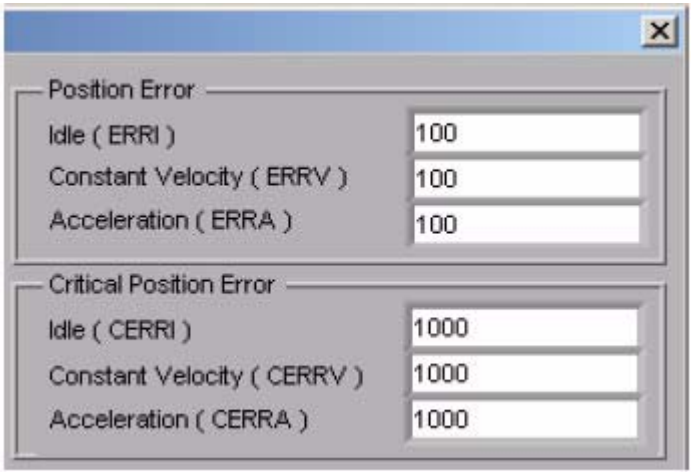
- Idle (motor not moving): **ERRI** (position error in idle state), **CERRI** (critical position error in idle state).
- Constant velocity: **ERRV**, **CERRV**
- Acceleration: **ERRA**, **CERRA**

For example, **w**hen the motor is in idle state, the criteria for determining whether there is a **PE** fault is **ERRI** and the criteria for determining whether there is a **CPE** fault is **CERRI**.

Note	
	*It is recommended to assign high following error fault thresholds for the duration of the configuration and adjustment process. This will accommodate the process while maintaining a measure of safety.* *Once the process is completed, the thresholds should be reduced to minimal levels.*

5.3.3.1 Setting Axis Following Error Thresholds

1. Open **MMI → Setup → Adjuster**.
2. Select the axis you want to configure and click **Safety Parameters**.
3. For both the **Position Error** and the **Critical Position Error**, enter the maximum acceptable error (in user units) for each of the motor states.



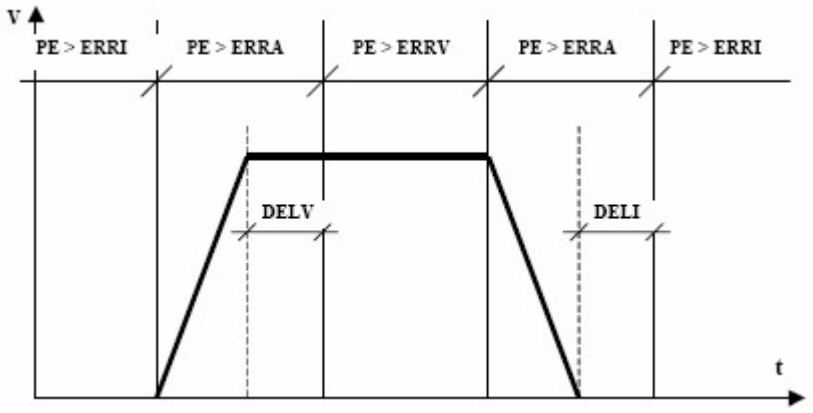
5.3.4 State Transition Delays for Following Error Thresholds

If required, delays can be configured to cause the controller, after changing motor state, to wait before it starts to check whether the following error has exceed the threshold associated with the new state.

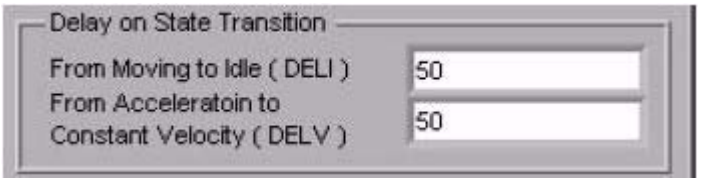
The controller provides two delay variables:

- **DELI** - Time (milliseconds) to wait on transition from acceleration state to idle state
- **DELV** - Time (milliseconds) to wait on transition from acceleration to constant velocity state.

The following diagram illustrates the use of these delays when checking for a PE fault for a typical motion profile:



In the **Safety Parameters** window, enter the **DELI** and **DELV** values.

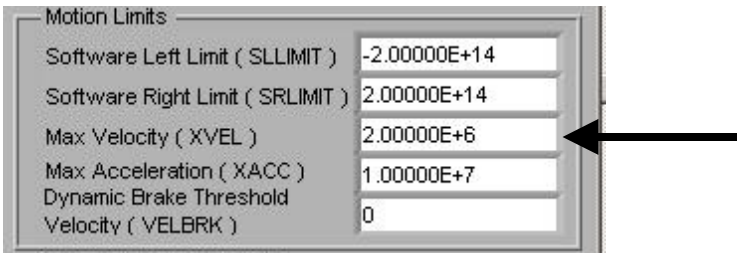


5.3.5 Maximum Velocity And Acceleration

The maximum velocity and acceleration that the controller is allowed to generate can be defined per axis in user units. If these limits are exceeded, a fault is generated.

5.3.5.1 Setting Axis Velocity And Acceleration Limits

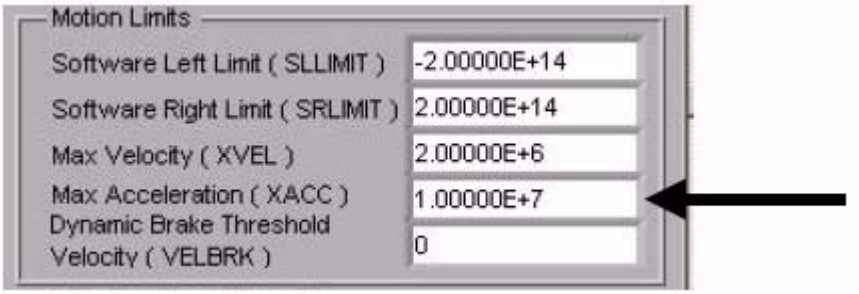
1. Open **MMI → Setup → Adjuster**.
2. Select the axis and click **Safety Parameters**.



3. Enter the **Maximum Velocity (XVEL)** for the axis.

Caution	
	***The maximum velocity variable (XVEL) is used as a scale factor for other adjustment variables. Therefore, it is important to define a proper value for it. Defining a value that is not reasonable for the feedback characteristics and required velocity of your application will lead to poor adjustment results.***

4. Enter the **Maximum Acceleration** (**XACC**) for the axis.



5.3.6 Axis Maximum Current (Torque)

The controller supports the following current fault and limits (per axis):

- Overcurrent fault (**CL** bit): as defined by the maximum allowed RMS (**XRMS**) which is calculated for the length of time defined by the RMS time constant (**XRMST**).
- **XCURI** limit: Maximum allowed current when the motor is idle.
- **XCURV** limit: Maximum allowed current when the motor is moving.

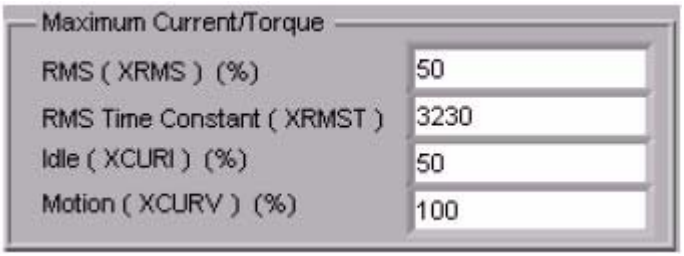
Note	
	*When working with third party drives the controller converts the current command to 10V ptp.*

The current limits are defined as a percentage of the maximum current (torque) command.

Caution	
	***If the drive is capable of generating higher peak or continuous current than the motor is rated for, the current limits must be set correctly to prevent damage to the motor.***

5.3.6.1 Setting Axis Maximum Current/Torque Limits

1. Open **MMI → Setup → Adjuster**.
2. Select the axis and click **Safety Parameters**.



3. Enter the **Idle** (**XCURI**) limit. It is recommended to set this value at the minimum current level required by the motor to keep the axis in position. The value should not exceed the **XCURV** value.
4. Enter the **Motion** (**XCURV**) limit state according to the formula: **XCURV** = [allowed motor peak current / drive peak current] x 100. Refer to the drive and motor specifications for these peak values.
5. Enter the **RMS** (**XRMS**) limit according to the formula: **XRMS** = [allowed motor nominal current / drive peak current] x 100. Refer to the drive and motor specifications for these values.
6. Enter the **RMS Time Constant** in msec (**XRMST**). This is the time interval for checking whether an overcurrent fault has occurred (meaning that XRMS has been exceeded).

The overcurrent fault activation time is according to the following formula:

fault activation = -ln(1-(XRMS/DCOM)^2)*XRMST

For example: if XRMS=50, DCOM = 100 and XRMST = 3300 msec

fault activation = -ln (0.75)*3300 = 950msec

Example: Setting current (torque) limits:

- **Drive specification**: peak current 20A, nominal (continuous) 10A.
- **Motor specification**: peak current 10A, nominal (continuous) 3A.

In this case:

- **XCURI** = minimum current level required by the motor to keep the axis in position, and less than **XCURV** (which is 50%).
- **XCURV** = (10/20)*100 = 50%.
- **XRMS** = (3/20)*100 = 15%.

5.3.7 Safety Faults Verification

1. Open **MMI → Safety Monitor**.
2. Verify that all fault indicators are green, indicating that no fault is present.

For faults with dedicated inputs, the **Safety Monitor** shows the state of both the input and the fault. The logic of the fault state is determined by the related bit in variable **SAFINI** or **S_SAFINI**. Depending on the bit state, the fault state is determined to be either the associated input state or the inverse of the input state.

For example, in the **Safety Monitor** window, shown in **Figure 17**, the **Emergency Stop Input** is red, indicating the physical input is high. However, the **Emergency Stop Fault** is green (fault not active). This result occurs because the **S_SAFINI.#ES** bit has been set, causing the controller to invert the logic of the Emergency Stop input.

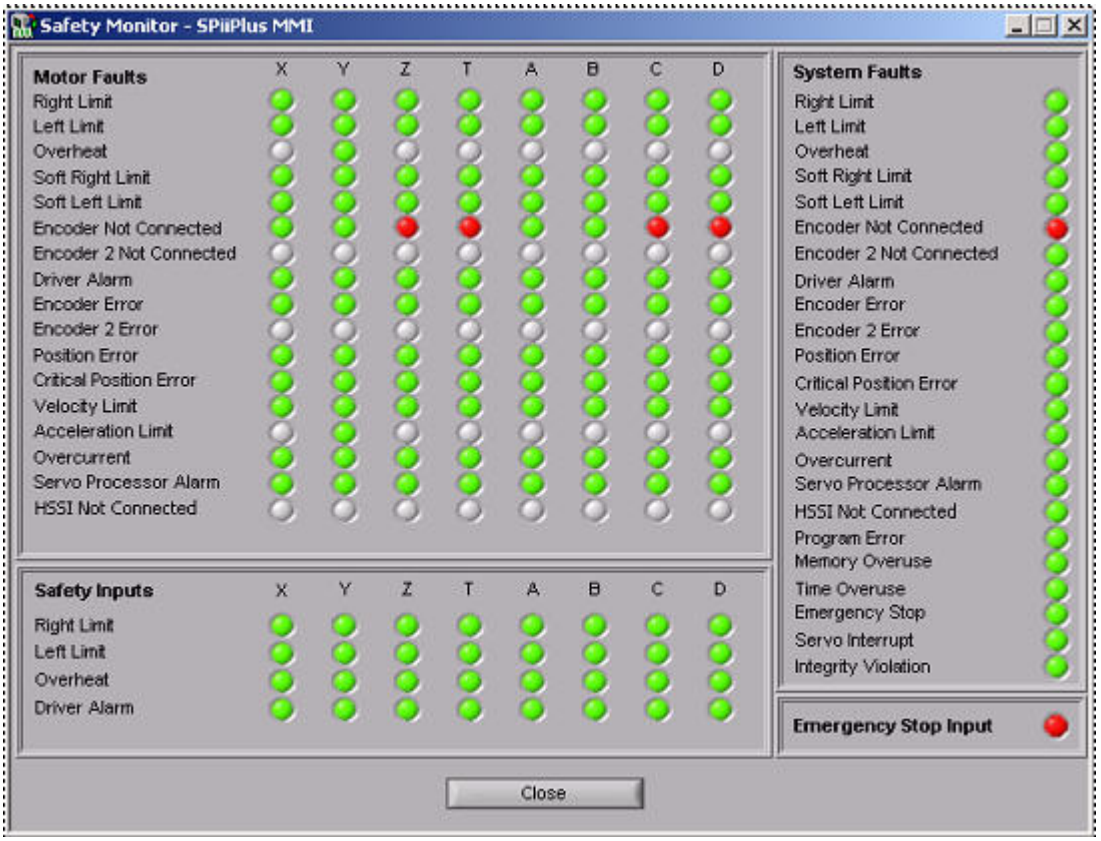


Figure 17 Safety Monitor Example with Inverted Emergency Stop Input

5.4 Mechanical Brake

For each axis the controller can activate a mechanical brake which has a normally applied logic. Typically, the mechanical brake is used to keep the position of the axis while the drive is disabled.

The controller provides a mechanism that automatically activates the brake when the drive is disabled and deactivates the brake when the drive is enabled.

Model	
	*SPiiPlus PCI: Brake activation is done via a digital output that is reconfigured to perform that task.* *SPiiPlus CM: Brake activation is done via a dedicated brake output without any need for configuration.*

In addition, you can define delays for these operations. For example, you may want the controller to wait 50ms once the drive is enabled before releasing the brake.

Refer to the *ACSPL+ Command & Variable Reference* for enable command timing diagrams using positive and negative BOFFTIME values, and disable command timing diagrams using positive and negative BONTIME values.

Caution	
	***The controller is designed to work with a normally-applied brake: meaning that the brake will be released when the signal is 1.***

5.4.1 Setting Up a Mechanical Brake

1. Open **MMI → Setup → Configurator**.
2. Select the axis and click **Motor Flags**.
3. In the **Motor Flags** (**MFLAGS**) dialog box, select the **Mechanical Brake** bit (23). This configures the controller so that brake activation/deactivation will be triggered by a drive **disable**/**enable** command.

4. Enter a value for the **Brake Off Time** (**BOFFTIME**). This value determines how long to wait (milliseconds) after enabling the drive before deactivating the brake.
5. Enter a value for the **Brake On Time** (**BONTIME**). This value determines how long to wait (milliseconds) after activating the brake before disabling the drive.

6. SPiiPlus PCI only: It is necessary to configure a digital output as the brake output. The output will carry the command to activate (output signal: 0) or deactivate (output signal: 1) the brake. The output will not be available for other digital output purposes while assigned to the brake. The command syntax is:

   setconf (29, *output number*, 2)

where *output number* is axis-specific (see **Table Table 4. Digital Outputs for Brake Control in SPiiPlus PCI**, **Table Table 5. Digital Outputs for Brake Control in SPiiPlus CM**, **Table Table 6. Digital Outputs for Brake Control in SPiiPlus SA, 3U, 3U-HP, 3U-LT & SAR-LT** and **Table Table 7. Digital Outputs for Brake Control in MC4U, MC4U-HP, & SPiiPlus LF**). For example, the command **setconf (29, 8, 2)** assigns output 8 to control the C axis mechanical brake.

This command can be sent to the controller via: **MMI → Terminal** or it can be included in a host or controller program.

Note	*This output assignment must be declared every time the controller is powered up.*

Table 4 Digital Outputs for Brake Control in SPiiPlus PCI

Digital Output	Controls Mechanical Brake for Axis	Digital Output	Controls Mechanical Brake for Axis
OUT0.0	X	OUT0.6	B
OUT0.1	Y	OUT0.7	-
OUT0.2	Z	OUT0.8	C
OUT0.3	-	OUT0.9	D
OUT0.4	T	OUT0.10	-
OUT0.5	A	OUT0.11	-

Table 5 Digital Outputs for Brake Control in SPiiPlus CM

Digital Output	Controls Mechanical Brake for Axis
OUT1.0	X
OUT1.1	Y
OUT1.4	A
OUT1.5	B

Table 6 Digital Outputs for Brake Control in SPiiPlus SA, 3U, 3U-HP, 3U-LT & SAR-LT

Digital Output	Controls Mechanical Brake for Axis
OUT0.0	X
OUT0.1	Y
OUT0.2	Z
OUT0.3	T
OUT0.4	A
OUT0.5	B
OUT0.6	C
OUT0.7	D

Table 7 Digital Outputs for Brake Control in MC4U, MC4U-HP, & SPiiPlus LF

Digital Output	Controls Mechanical Brake for Axis
OUT0.0	X
OUT0.1	Y
OUT0.2	Z
OUT0.3	T
OUT0.4	A
OUT0.5	B
OUT0.6	C
OUT0.7	D

7. Open the **MMI → Terminal**.
8. Send the following command line: `enable <axis>` where `<axis>` is the name of the axis (X, Y, etc.).
9. Verify that the mechanical brake is deactivated by the command.

Verify that mechanical brake is activated by the command line:
`disenable <axis>`

Model	
	*To set a digital output to work as a general purpose output after it was defined to control a mechanical brake:* • *In SPiiPlus PCI: In the* **MMI → Configurator → Motor Flags** *(***MFLAGS***) dialog box, unselect the* **Mechanical Brake** *bit (23). Send the* ***setconf*** *(29, output number, 0) command.* • *In SPiiPlus CM or SA: In the* **MMI → Configurator → Motor Flags** *(***MFLAGS***) dialog box, unselect the* **Mechanical Brake** *bit (23).*

5.5 Virtual Axis

One or more of the SPiiPlus axe can be defined as a virtual axis. A virtual axis is normally not connected to any hardware. Instead it is used as a master axis for one or more slaved axes (connected to real motors). Each slave axis can be related to the virtual axis with a user-defined gear ratio.

5.5.1 Configuring a Virtual Axis

1. To define a virtual axis, run a program with the following ACSPL+ commands:

```
MFLAGS<axis number>.17=0                ! Allow non-default connect command
CONNECT RPOS<axis number>.=0            ! Define axis as virtual axis
STOP                                    ! End program
```

2. To re-define a virtual axis as a normal axis, run a program with the following ACSPL+ commands:

```
MFLAGS<axis number>.17=0                ! Allow non-default connect command
CONNECT RPOS<axis number>.=APOS>axis number! Define axis as normal axis
STOP                                    ! End program
```

Note	
	*Do not mix a virtual axis with a dummy axis (MFLAGS bit 0). Setting an axis as a dummy will:* 1. *Free both of the analog outputs of the axis for general purpose use.* 2. *ENABLE command for the axis drive is blocked.*

5.6 Single and Dual Loop Control

The controller supports two control feedback configurations per axis:

- **Single Loop** (default) – control of the axis is based on one feedback source.
- **Dual Loop** – control of the axis is based on two feedback sources.

Dual loop control is commonly used when there is poor stiffness between the motor and the load (for example, in belt-driven systems). Dual loop control for such applications does not suffer from the drawbacks of single loop control, such as reduced dynamic performance due to the poor stiffness and backlash between the motor and the load.

Examples of applications that benefit from dual loop control:

- Linear stage drive with a screw (linear encoder for stage position, rotary encoder for motor velocity and for brushless software commutation for a brushless motor).
- Printer drum drive with a flexible belt (rotary encoder for drum position, rotary encoder for motor velocity and for brushless software commutation for a brushless motor).
- A high-inertia antenna driven by a high-ratio gear motor (rotary encoder for antenna position, rotary encoder for motor velocity and for brushless software commutation for a brushless motor).

Dual loop control works as follows:

1. Load position feedback is used as input for the position loop.
2. Motor position feedback is used as input for the velocity loop.

Dual loop control can only be used for axes X, Y, Z, and/or T. In each case, a second axis is defined as a dummy axis, respectively: A, B, C or D. For example, if dual loop control is used for axis Y, then a dummy B axis must be defined as well. Therefore, dual loop control always consumes two of the total available controller axes.

Figure 18 illustrates a sample dual loop application. A feedback device (#1) is located on the motor shaft and used as input for the velocity loop. A second feedback device (#2) is located next to the load and used as input for the position loop.

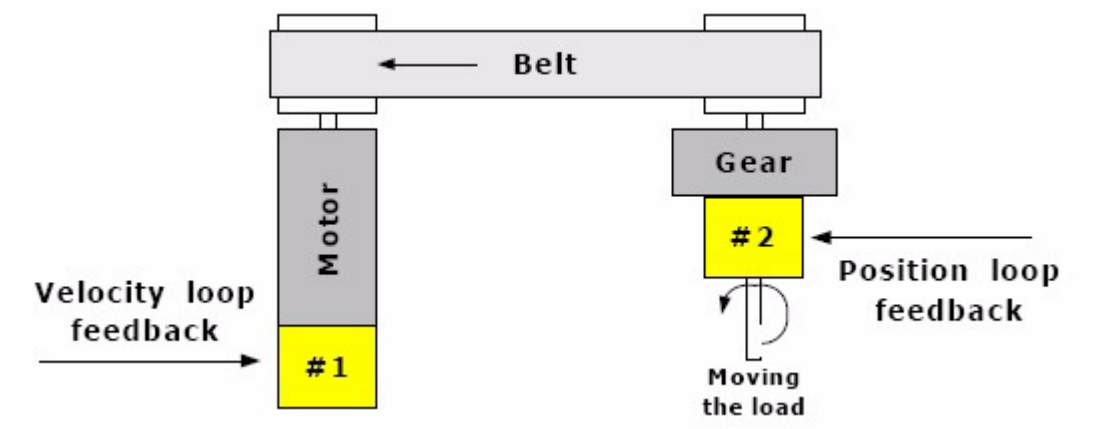


Figure 18 Dual Loop Control Example

5.6.1 Calculation the Gear Ratio

Before configuring the dual loop control, obtain the gear ratio between the velocity loop feedback and the position loop feedback. If the gear ration is unknown, find it according to the following procedure:

1. Open **MMI → Motion Manager**.
2. Select both controller axes that represent the velocity loop feedback and the position loop feedback (for example, X and A).
3. Click **Zero Feedback Position**.
4. Disable both axes.
5. Manually move the axis.
6. Check the position of the velocity loop feedback and of the position loop feedback.
7. Calculate the gear ration by dividing the position loop feedback value with the velocity loop feedback value.
8. Record this value for the dual loop control configuration.

5.6.2 Configuring the Dual Loop Control

1. Open **MMI → Setup → Configurator.**
2. Decide which axis will be the dual loop axis (X,Y,Z or T). Look up the associated dummy axis in **Table 8** and select the dummy axis in the **Configurator** window.

Table 8 Dual Loop Axis Pairs

Dual Loop Axis	Dummy Axis
X	A
Y	B
Z	C
T	D

3. Click **Motor Flags** (**MFLAGS**) to open the Motor Flags window.
4. Select **Dummy Axis** (**Bit #0**).
5. In the **Configurator** window, select the axis to be defined as dual loop.
6. Determine the user units for the position loop feedback by entering a value for the **Encoder Factor** (**EFAC**). The feedback resolution is that of the load position feedback device. See **Section 5.2.1, User Units of Measure (EFAC Variable)** on **Page 25** for an explanation of the **EFAC** formula.
7. Determine the user units for the velocity loop feedback by entering a value for the **Secondary Encoder Factor** (**E2FAC**) as follows:
   a) Open **MMI → Setup → Adjuster**.
   b) Select the dual loop axis.
   c) Open **Axis Setup → General** (tab).
   d) For **Control Configuration** select **Dual Loop**.
   e) Click the **Position Feedback** tab.
   f) Enter the value of the gear ratio as defined in the **"Calculation the Gear Ratio" on Page 44**.
   g) Click the **Velocity Feedback** tab.
   h) Define the feedback device variables (per specifications of the motor position feedback device) – see **Section 5.10**, **Feedback Devices** on **Page 54**.
   i) Click the **Position Feedback** tab and define feedback device variables.

Note

In single loop control, the feedback position is represented by the **FPOS** *variable.*

In dual loop control, the load (position loop) feedback position is represented by **FPOS** *and the motor (velocity) feedback position by* **F2POS**.

For example, in the **MMI → Terminal**, *the command:*

```
?X_FPOS,?X_F2POS
```

will display the two feedback values for the X axis.

5.7 Servo Drives (DC Brush, DC Brushless)

The controller provides two options for interfacing with an external servo drive:

- **Remote (HSSI) connection**: drive commands are sent via an HSSI channel (a high speed serial communications link implemented over Ethernet-type cable) to an interface module (like the HSSI-ED2) or to a drive module (like the HSSI-SA-1).
- **Direct connection**: one (or two depending on the type of motor) 10V ptp drive commands are sent directly to the drive.

Model 	*SPiiPlus CM includes integrated digital drives with fixed axis assignments. For these drives, the* **Drive** *tab in the* **Axis Setup** *window is irrelevant (because the values are already defined in the controller).*

5.7.1 Setting Up a Remote (HSSI) Servo Drive Connection

Each SPii processor supports one HSSI channel. The number of SPii processors in the controller depends on the controller model (refer to the controller's hardware guide). Each HSSI channel is divided into four registers. The number of registers consumed by the HSSI module(s) connected to the channel, depends on the module type, see**Table 9**.

Table 9 HSSI Modules and HSSI Channel Resources Consumed

HSSI Module	Description	Registers Consumed
HSSI-ED2	Interface for two distributed axes	4
HSSI-IO16	Additional 16 digital inputs and 16 digital outputs	1
HSSI-SA-1	Distributed servo drive	2

The HSSI channel number and register that the controller allocates to the drive command depend on the axis, as described in **Table 10**.

Table 10 HSSI Axis Allocations

Axis	HSSI (channel:register)		Axis	HSSI (channel:register)
X	HSSI 0:0		A	HSSI 0:2
Y	HSSI 1:0		B	HSSI 1:2
Z	HSSI 2:0		C	HSSI 2:2
T	HSSI 3:0		D	HSSI 3:2

Set the drive connection as follows”

1. Open **MMI → Setup → Adjuster**.
2. Select the axis and click **Axis Setup → General** (tab).
3. For the **Drive Connection**, select **Remote**.

Warning	
	***When the HSSI Manager window is open, the controller loses communication with all the HSSI modules.*** ***Before opening the HSSI Manager, disable all HSSI remote axes.***

4. Open the **Tools** (menu) **→ HSSI Manager**. This window is used to establish communication between the controller and the module(s) at the other end of each HSSI communication channel.

5. Click **Reconnect.** The controller automatically detects all connected HSSI modules and starts communicating with them.

5.7.2 Setting Up a Direct Connected Servo Drive

The controller supports three types of third party drives, described in **Table 11**, for servo (DC brush and DC brushless) motors.

Model	
	*SPiiPlus PCI-4/8 does not support drives with single-ended current command inputs.*

Table 11 Third Party Supported Drives

		SPiiPlus Product
1.	DC brush drive having one 10V ptp differential current command input.	SPiiPlus PCI - up to eight exes SPiiPlus CM via HSSI-ED2 – up to seven exes
2.	Three-phase DC brushless drive having two 10V ptp differential current command inputs. The controller performs the commutation.	SPiiPlus PCI - up to eight exes. SPiiPlus CM- none (can be done by integrated drives only)
3.	Three-phase DC brushless drive having one 10V ptp differential current command input. The drive performs the commutation.	SPiiPlus PCI - up to eight exes. SPiiPlus CM via HSSI-ED2 – up to seven exes.

1. Open **MMI → Setup → Adjuster**.
2. Select the axis and click **Axis Setup → Drive** (tab).
3. In the **Axis Setup** window, from the **Edit Database** menu, select **Drive → Database Name**. Create a new drive database file. This database file is contain the data of all the axes of the machine.
4. In the **Drive** tab fields select/enter the following:
   a) **Name**: drive manufacturer and model name.
   b) **Type**: Select from the list.
   c) **DC Bus Voltage [V]**: refer to drive's specifications.
   d) **Nominal Current [A]**: refer to drive's specifications.
   e) **Peak Current [A]**: refer to drive's specifications.
   f) **Description**: other relevant information.
5. From the **Edit Database** menu, select **Drive → Add Item** to save the new drive description.
6. Open **MMI → Setup → Configurator**.
7. Enter a value (milliseconds) for **Enable Time** (**ENTIME**). This field determines the interval allowed between when the drive enable signal is sent and when the drive needs to be successfully enabled. Refer to the drive's specifications to determine this value.

5.8 Servo (DC Brush, DC Brushless) Motors

The controller supports two types of servo motors:

- DC brush motor.
- Three-phase DC brushless motor.

5.8.1 Setting Up a Servo Motor

1. Open **MMI → Setup → Adjuster**.
2. Select the axis and click **Axis Setup**.
3. Select the **Motor** tab.
4. From the **Edit Database** menu select **Motor → Database Name**.
5. Create a new motors database file. The same database file is used for all the axes.
6. In the **Motor** tab fields, select/enter the following:
   a) **Name**: motor manufacturer and model name.
   b) **Topology**: **Rotary** or **Linear**.
   c) **Type**: **DC Brush** or **DC Brushless**
   d) **Nominal Current [A]**: refer to motor's specifications.
   e) **Description**: other relevant information.
7. In **Edit Database** menu select **Motor→ Add Item** to save the new motor description.

5.9 Step Motor Drives

Note	
	*The controller's PEG output and Mark input features (see descriptions in SPiiPlus ACSPL+ Programmer's Guide) are only available for stepper motors if at least one of the following bits is set:* **MFLAGS.5** *or* **MFLAGS.6**.

The controller supports drives for pulse–direction (P-D) step motors. Up to half the number of axes supported by the controller can be step motors. Only the A, B, C and D axes can be used for step motor control.

Step motor control is open loop only. However a feedback device can be used with a step motor for reading the position (it will not affect the control of the motor).

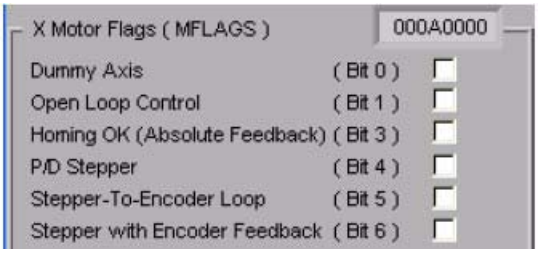
The configuration of **MFLAGS** bits 4, 5, and 6 determines the step motor feedback configuration.

Table 12 Step Motor Feedback Configurations

MFLAG.4	MFLAG.5	MFLAG.6	Feedback	Description
1	0	0	None	• open-loop control, • no encoder installed, • **FPOS** always equal to **RPOS**
1	1	0	Stepper To Encoder Loop	• open-loop control, • no encoder installed, • **FPOS** always equal to **RPOS** • stepper output internally wired to encoder input This configuration is used mostly to enable the encoder interface features (Mark, PEG). PEG and MARK are based in this case on **RPOS** and not on the exact feedback position, which is unknown.
1	0	1	Encoder Feedback	• open-loop control • encoder installed • **FPOS** equal to actual encoder position When the motor is disabled, **RPOS** is equal to **FPOS** (like for any type of motor). Once the motor is enabled, the controller starts with RPOS=FPOS, but then the control continues in open loop. If eventually a difference between RPOS and FPOS appears, the controller never corrects it automatically. However, the user application can verify the difference and introduce correction if required.

5.9.1 Setting Up a Step Motor Drive

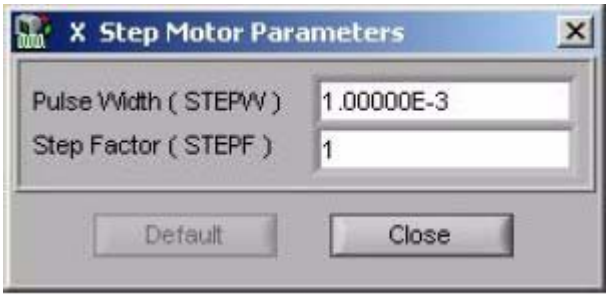
1. Open **MMI →Setup → Configurator.**
2. Select the axis name (A, B, C or D only).
3. Click **Motor Flags (MFLAGS)** to open the **Motor Flags** window.



4. Select bit 4 and bit 5 or 6 as follows:

- **P/D Stepper** (bit 4): defines the axis as a step motor.
- **Stepper-To-Encoder Loop** (bit 5): enables the pulses of the step motor commands to be read as if they were encoder position input – by the position feedback (**FPOS**) variable of the stepper axis.
- **Stepper With Encoder Feedback** (bit 6): enables encoder feedback from the step axis to be read to the position feedback (**FPOS**) variable of the stepper axis. This will not affect the control of the motor (position error remains zero).

5. Open **MMI → Setup → Configurator → Step Motor Parameters**.



6. Define the **Pulse Width (STEPW)** per the step drive specifications. Verify the Pulse Width length per the following equation:

   For a rotary axis:

$$STEPW < \frac{\frac{1000}{(\text{Maximum planned velocity (RPM)}/60)}}{\text{Number of SPiiPlus pulses to make one rotation}}$$

For a linear axis:

$$STEPW < \frac{\frac{1000}{\text{Maximum planned velocity (meter/sec)}}}{\text{Number of SPiiPlus pulses to move 1 meter}}$$

7. Define the **Step Factor (STEPF)** as follows:
   a) Take the number of steps of the stepper motor (for example 200 per rotation).
   b) Take the multiplication factor of your stepper (or micro-stepper) drive (for example x256).
   c) Determine the number of SPiiPlus pulses required to make one rotation (steps x factor) or move one user unit. In the example cited in steps A and B, this would be 51,200 pulses (200x256=51,200).
   d) Define **STEPF** as follows:

      For a rotary axis:

$$STEPF = \frac{\text{Number of user units per rotation}}{\text{Number of SPiiPlus pulses to make one rotation}}$$

      For a linear axis:

$$STEPF = \frac{\text{Required user unit (like mm, cm)}}{\text{Number of SPiiPlus pulses to move one user unit}}$$

Note the following three examples:

- Example 1 (rotary motor):
  - User unit per rotation: default (**EFAC**=1)
  - Number of motor steps: 200 steps
  - Step drive multiplication factor: x256
  - Number of SPiiPlus pulses to make one rotation: 200x256 = 51,200
  - Required **STEPF**: 51,200/51,200 = 1

Remark: In this case a command of `PTP/R,51,200` will cause one rotation of the motor.

- Example 2 (rotary motor):
  - User unit per rotation: degree (360 per rotation)
  - Number of motor steps: 200 steps
  - Stepper drive multiplication factor: x256

- Number of SPiiPlus pulses to make one rotation: 200x256 = 51,200
- Required **STEPF**: 360/51,200 = 0.0070312

Remark: in this case a command of `PTP/R,360` will cause one rotation.

- Example 3 (linear axis):
  - User unit: mm
  - Number of steps of the stepper motor per mm: 200 steps
  - Stepper drive multiplication factor: x1 (no multiplication).
  - Number of SPiiPlus pulses to move one mm: 200
  - Required **STEPF**: 1/200 = 0.005

Remark: in this case a command of `PTP/R,1` will cause a 1mm movement.

Warning 	***If you select the* Stepper *with* Encoder Feedback *option, verify that the direction of motion matches the direction of the feedback counting. You can invert feedback counting using Invert Encoder Direction, located at* MMI → Setup → Adjuster → Open Loop Verification → Feedback Configuration → Invert Encoder Direction.**

5.9.2 Correcting Stepper Position Using Feedback

Advanced 	*This topic is for users who are familiar with SPiiPlus ACSPL+ programming.*

When the stepper axis is equipped with a feedback device, the controller can correct the position reached at the end of each motion to the desired position.

Such an implementation can be done by running the following program continuously:

```
! This program corrects the stepper target position by feedback
! Change the stepper position by the DesiredPosition variable
GLOBAL INT DesiredPosition !Declare variable called DesiredPosition
INT STEPPER_TARG_RAD       !Declare variable called STEPPER_TARG_RAD
INT STEPPER_TARG_RAD       !Declare variable called STEPPER_TARG_RAD
INT AXIS                   !Declare variable called AXIS
AXIS=1                     !Define AXIS number (A=0, B=1...).
STEPPER_TARG_RAD=2
WHILE 1                    ! Run forever
TILL (MST(AXIS).#ENABLED=1 & MST(AXIS).#MOVE=0)
                           ! Wait till the drive is enabled
IF ABS(DesiredPosition-FPOS(AXIS))>STEPPER_TARG_RAD
                           !Check the final position
PTP/RE (AXIS), DesiredPosition-FPOS(AXIS)
                           !Correct the position
END
END
```

In this case, change the **DesiredPosition** value to initiate the stepper motion.

5.10 Feedback Devices

The controller supports both digital and analog feedback.

The following digital incremental encoders (with differential lines) are supported:

- a quad b (quadrature) & index
- clock-dir & index
- up–down & index

The following analog feedback devices (with differential lines) are also supported:

- sin-cos & index incremental encoder (ordering option)
- other feedback device via controller's analog input

Note	
	*The controller also supports Netzer Precision absolute encoders, see* **Chapter 14 - "Appendix B: Netzer Precision Electric EncoderS™"** .

5.10.1 Encoder Input Clock

Advanced 	*This topic is for users who are familiar with SPiiPlus ACSPL+ programming and is provided for information only.*

The encoder is represented in the controller as a synchronous state machine that is activated by a clock, with programmable frequency in the SPii processor:

Table 13 Programmable encoder sampling rates

E_FREQ variable	Encoder maximum sampling rate
2	5MHz
20	40MHz
60	120MHz

Note 	1. *The clock frequency (***E_FREQ***) can be selected for the X, Y, Z, and T axes only. Each of these axes is paired in the controller with a second axis (X with A, Y with B, Z with C, and T with D).* *When the frequency is set for the primary axis (X, Y, Z, T), the same frequency is automatically applied to its secondary axis.* 2. *For a Sin-Cos encoder, the clock frequency (***E_FREQ***) is automatically set to 2 (5MHz) which applies automatically to the related secondary axis as well.*

The maximum rate of A quad B counting, pulse-direction counting, and up-down counting that the SPiiPlus can handle is equal to half the frequency of the clock. This means that the maximum rate of a digital encoder channel can be 60MHz, 20MHz (default) or 2.5MHz.

The encoder signals are filtered by a three-stage digital filter. The higher the clock frequency, the less filtering is done. For a 5 MHz clock, an input signal that is not stable for at least 600 nanoseconds (3 clock cycles), is considered to be noise and is ignored. For a 120MHz clock, signals that are not stable for 25 nanoseconds are considered noise and ignored.

An edge separation between the two encoder input channels must be at least 1 clock cycle. If the separation is less, it is considered a simultaneous transition, which is results in a miscount and an encoder fault.

Table 14 Recommended Digital Encoder Input Frequencies (E_FREQ)

Maximum Frequency	Recommended E_FREQ
up to 2 million counts/sec	2 (5MHz)
2 to 12 million counts/sec	20 (40MHz)
12 million counts /sec and higher	60 (120MHz)

The frequency of the encoder clock also affects Position Event Generator (PEG) accuracy. PEG is a controller feature where position events trigger output pulses with sub microsecond delay.

Table 15 E_FREQ and PEG Delay

Clock Frequency (MHz)	PEG Delay
2	< 1.27microsec
20	< 240nsec
60	< 115nsec

5.10.2 Encoder Timing Limitations

Figure 19–Figure 21 illustrate the following timing limitations

- A Quad B Timing
- Pulse – Direction Timing
- Up – Down Timing

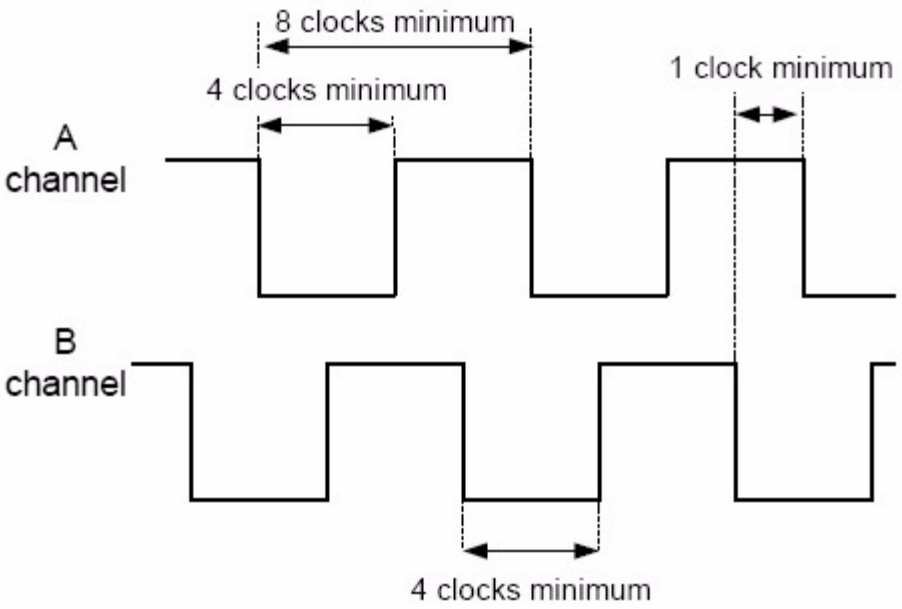


Figure 19 A Quad B Timing

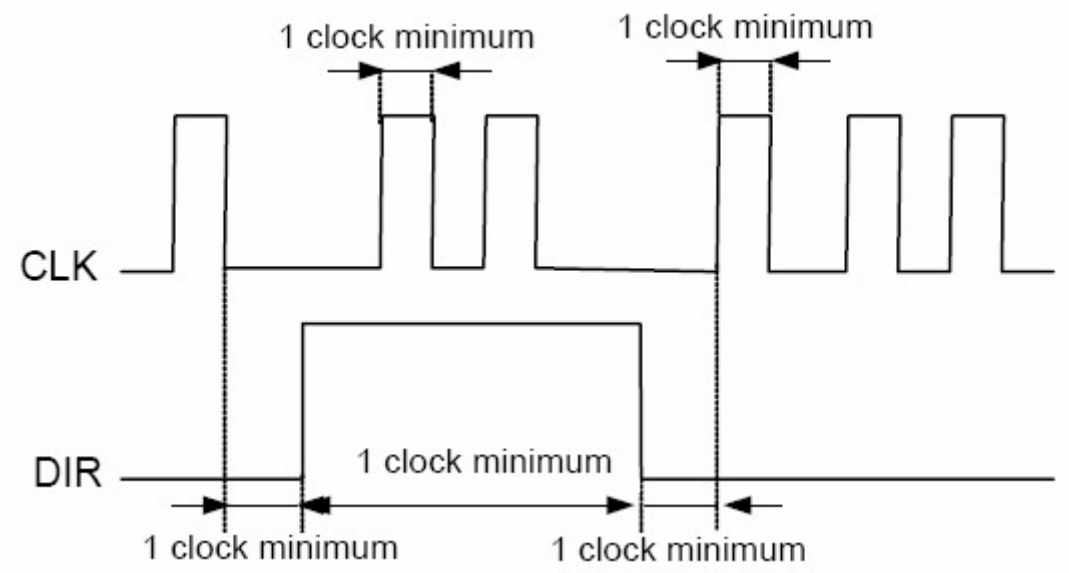


Figure 20 Pulse-Direction Timing

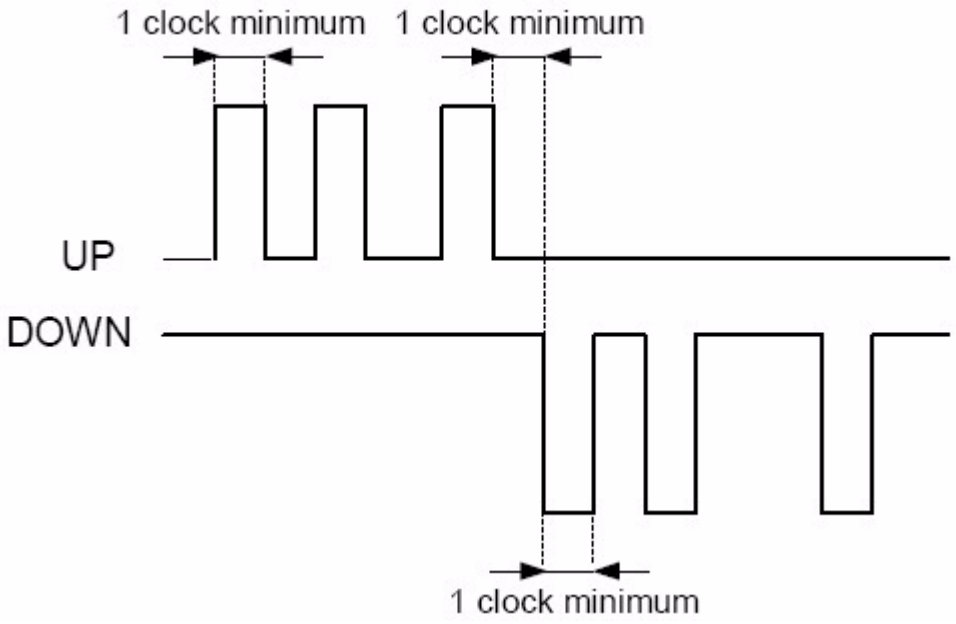


Figure 21 Up-Down Timing

5.10.3 Configuring Digital Encoder Feedback

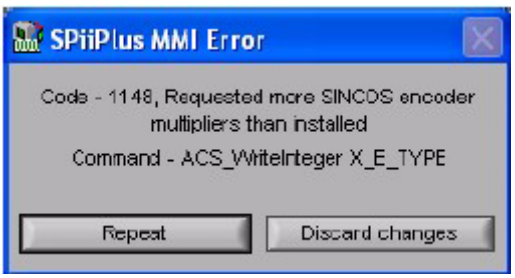
1. Open **MMI → Setup → Adjuster**.
2. Select the axis and click **Axis Setup**.
3. Click the **Position Feedback** tab (single loop control) or **Velocity Feedback** tab (dual loop control).
4. From the **Edit Database** menu, select **Encoder → Database Name**. Create a new encoder database file. The same database file is used for all the axes.
5. In the Position/Velocity Feedback tab fields select/enter the following:
   a) **Name**: encoder manufacturer and model name.
   b) **Topology**: **Rotary** or **Linear**.
   c) **Type**: select **Quadrature, Up-Down,** or **Pulse-Direction**.
   d) **Encoder Frequency**: select (for X, Y, Z and T axes only) according to the **Encoder Timing Limitations**.
   e) ***Lines Per Revolution** (*if **Topology** value is **Rotary)**: Enter a value or select from the list. According to the encoder's specifications.
   f) ****Lines** (**if **Topology** value is **Linear)**: Select the units: **per mm** or **per inch** then enter a value or select from the list –refer to the encoder's specifications.
   g) **External Multiplier**: If no external multiplier is necessary, select 1 (default). If an external multiplier is used, select the multiplication factor.
   h) **Internal Multiplier**: Always 4 for a digital encoder.
   i) ***Counts per revolution** (*if **Topology** value is **Rotary)**: The product of the **Lines Per Revolution** x **External Multiplier** x **Internal Multiplier**.
   j) ****Counts per mm/inch** (**if **Topology** value is **Linear)**: The product of the **Lines** x **External Multiplier** x **Internal Multiplier**.
   k) **Description**: other relevant information.
6. From the **Edit Database** menu, select **Encoder→ Add Item** to save the new encoder description.

5.10.4 Configuring Feedback: Sin-Cos

Model	
	*SPiiPlus PCI requires jumper configuration for sin-cos encoder operation. Refer to the SPiiPlus PCI Hardware Guide.*

1. Open **MMI → Setup → Adjuster**.
2. Select the axis and click **Axis Setup**.
3. Click the **Position Feedback** tab (single loop control) or **Velocity Feedback** tab (dual loop control).

4. From the **Edit Database** menu, select **Encoder → Database Name**. Create a new encoder database file. The same database file is used for all the axes.
5. In the Position/Velocity Feedback tab fields select/enter the following:
   a) **Name**: encoder manufacturer and model name.
   b) **Topology**: **Rotary** or **Linear**.
   c) **Type**: Select **Sin-Cos**.

Note	*Support for sin-cos feedback is optional and must be included in the product order. If the controller doesn't support sin-cos feedback, the following error message will be displayed when you click* **Calculate Parameters and Close**. 

   d) ***Lines Per Revolution** (*if **Topology** value is **Rotary)**: Enter a value or select from the list. According to the encoder's specifications.
   e) ****Lines** (*if **Topology** value is **Linear)**: Select the units: **per mm** or **per inch** then enter a value or select from the list –refer to the encoder's specifications.
   f) **Internal Multiplier Factor**: Select the multiplication factor to achieve the feedback resolution you require. A high multiplication factor can be used to achieve higher feedback resolution and better dynamic performance. However, the quality of the sin and cosine input signals are limiting factors. It is recommended to start with a low multiplication factor and increase it until the required resolution is achieved with minimal degradation in performance (for example, increased position error).

Note	*The controller supports sin-cos encoder input rates of up to 250,000 sine/cosine periods per second.*

   g) ***Counts per revolution** (*if **Topology** value is **Rotary)**: The product of the **Lines Per Revolution** x **Internal Multiplier**.
   h) ****Counts per mm/inch** (**if **Topology** value is **Linear)**: The product of the **Lines** x **Internal Multiplier**.
   i) **Description**: other relevant information.

6. From the **Edit Database** menu, select **Encoder→ Add Item** to save the new encoder description.

Note	
	*While having an interpolated position with a sin-cos encoder, the Index, PEG and MARK position will **NOT** be based on the exact interpolated position of the axis but only on fix x4 interpolated factor.*

5.10.5 Configuring Feedback: Analog Input

Other types of analog feedback devices can be connected to the controller via the controller's analog inputs. The axes are mapped to feedback via the analog inputs as follows:

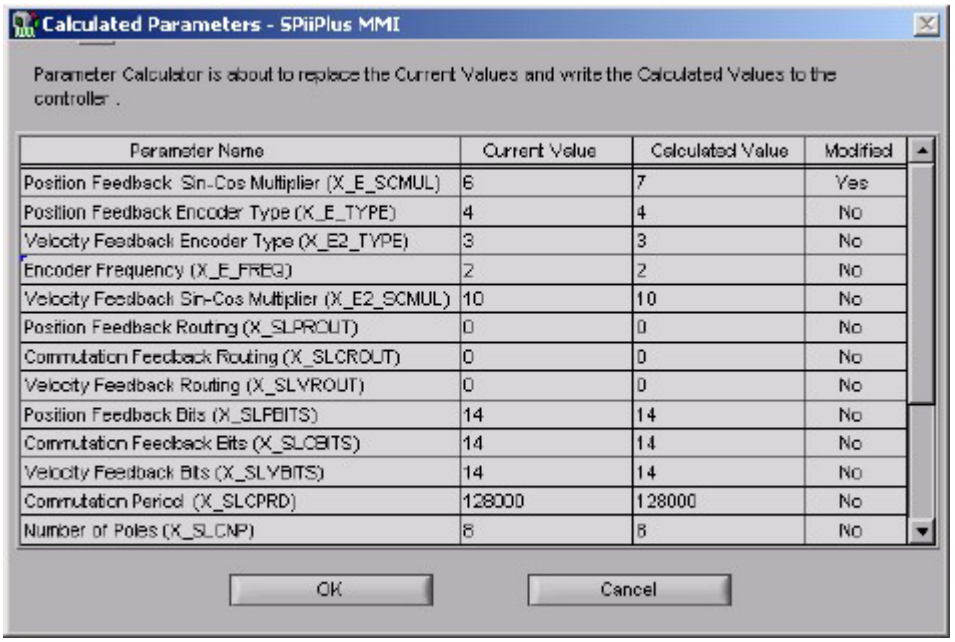
Table 16 Analog Feedback per Axis

Analog Input	Feedback for Axis
AIN0	X
AIN4	Y
AIN8	Z
AIN12	T
AIN2	A
AIN6	B
AIN10	C
AIN14	D

1. Open **MMI → Setup → Adjuster**.
2. Select the axis and click **Axis Setup**.
3. Click the **Position Feedback** tab (single loop control) or **Velocity Feedback** tab (dual loop control).
4. From the **Edit Database** menu, select **Encoder → Database Name**. Create a new encoder database file. The same database file is used for all the axes.
5. In the Position/Velocity Feedback tab fields select/enter the following:
   a) **Name**: encoder manufacturer and model name.
   b) **Topology**: **Rotary** or **Linear**.
   c) **Type**: select Analog Input
   d) ***Resolution**: Select the volts per units (*choices for /unit change depending whether **Topology** value is **Rotary** or **Linear)**. Then enter a value or select from the list –refer to the encoder's specifications.
   e) **Description**: for free text if needed
6. In **Edit Database** menu select **Encoder → Add/Replace Item**.

5.11 Calculating Parameters of Axis Control Loop, Motor, Drive and Feedback

1. After you have completed changes in the **Axis Setup** window, click **Calculate Parameters and Close**.
2. The **Calculate Parameters** window appears. This window lists all the involved configuration parameter. The new parameters that the user defined (and other calculated parameters based on the new definitions) will be saved in the RAM memory only after the OK button is pressed. For each parameter the “Current Value” shows the parameter value as exists in the RAM and the “Calculated Value” as will be saved to the RAM after the OK button is pressed. The “Modified” column shows if the parameter value was changed.
3. Click **OK** to save the changes to the RAM.



5.12 DC Brushless (AC Servo) Drive Bias

One of the motor-drive configurations that the controller supports is a three-phase DC brushless (AC servo) motor connected to an external drive with two drive command inputs (also known as a UV drive).

Theoretically, the drive should produce zero voltage in the three phases when the controller drive commands are zero. In a case where the drive outputs have a bias voltage when the controller commands are zero, this can be corrected with the **SLBIASA** and **SLBIASB** variables. If the external drive is a current drive, it must be connected to the motor while doing the bias adjustment. If it is a voltage drive, it doesn't have to be connected.

Note	*Bias adjustment for a of DC brush motor drive is also supported, see* ***Chapter 8 - "Open Loop and Index Verification"*** .

5.12.1 Adjusting Bias for a DC Brushless (AC Servo) Motor Drive

The purpose of this procedure is to adjust bias voltages that may exist in the controller **DAC** outputs or the drive analog inputs. The procedure should be repeated whenever a drive or a controller is replaced. The procedure assumes that a brushless motor is used and that the commutation is done by the controller.

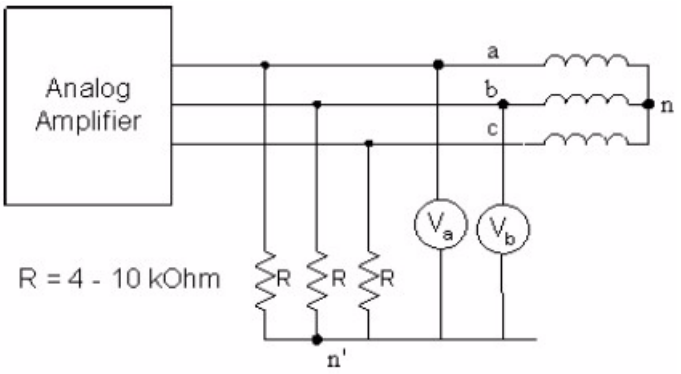
Theoretically, the drive should produce zero voltage in the three phases when the controller drive commands are zero. Practically, small biases exist in the controller DAC outputs and the drive analog inputs.

This can be corrected with the **SLBIASA** and **SLBIASB** variables. The two variables are specified in percentages of the maximal **DAC** command.

- **SLBIASA** is used to adjust the offset of phase "A "
- **SLBIASB** is used to adjust the offset of phase "B".

1. Verify that the drive is connected to the controller.
2. Verify that the drive is connected to the motor.
3. Set the commutation bit to **0** (**MFLAGS** bit **9**).
   For example: **X_MFLAGS.9 = 0** (for X axis)
4. Write down the existing values of **SLIOFFS** or **DCOM** parameters.
   In certain cases (like vertical axis) they may have non-zero values that are used to compensate gravity. (You can query them from the terminal, for example **? X_DCOM, X_SLIOFFS**)
5. Set **SLBIASA**, **SLBIASB**, **DCOM** and **SLIOFFS** to zero. The voltages on the motor phases "A" and "B" have to be measured

6. At the same time, the variables **SLBIASA**, **SLBIASB** have to be adjusted to bring both readings as close to zero as possible. This can be done by a single voltmeter (or multimeter) as follows:
   a. Enable the axis.
   b. Make sure that the variable **DOUT** shows zero value (you can query it from the terminal, for example **?X_DOUT**)
   c. Connect the voltmeter between phase “A” and the motor neutral point. Using the Communication Terminal, adjust **SLBIASA** to make the reading as close to zero as possible. A typical range is ±10 %. The value can be either positive or negative (for example: **X_SLBIASA = -5**).
   d. Connect the voltmeter between phase “B” and the motor neutral point. Using the Communication Terminal, adjust **SLBIASB** to make the reading as close to zero as possible.
7. In case the neutral point is not accessible, or the motor is connected in delta, the user may create an “artificial” neutral point by connecting three-phase resistors in parallel to the motor. The 3 resistors should be identical, in the range 4-10 kΩ. The user should then measure relative to the artificial neutral as shown in the following drawing:



8. Sometimes distinguishing between phase “A” and phase “B” may be a problem, e.g., the phases might be specified with different letters, or not be consistent with the controller definition. In this case the user can easily identify the phases:
   a. Phase "A" bias is only affected by **SLBIASA** variable. **SLBIASB** should not affect it (or have very minimal effect).
   b. Phase "B" bias is only affected by **SLBIASB** variable. **SLBIASA** should not affect it (or have very minimal effect).
   c. Phase "C" will be affected by both variables.
9. After setting the bias variables correctly, set **DCOM** or **SLIOFFS** according to their original values.
10. Save to FLASH.

6 Current Loop Adjustment

Model	
	*SPiiPlus CM: Current loop adjustment is required for the integrated digital drives of the SPiiPlus CM.* *For all other drive types, there is no current loop adjustment and this chapter is irrelevant.*

6.1 Overview

SPiiPlus CM control modules comprise a controller and integrated digital drives. The current control of these internal drives must be adjusted by the user.

6.2 Adjusting an Axis Current Loop

1. Open **Setup → Adjuster → Current Loop Adjustment**

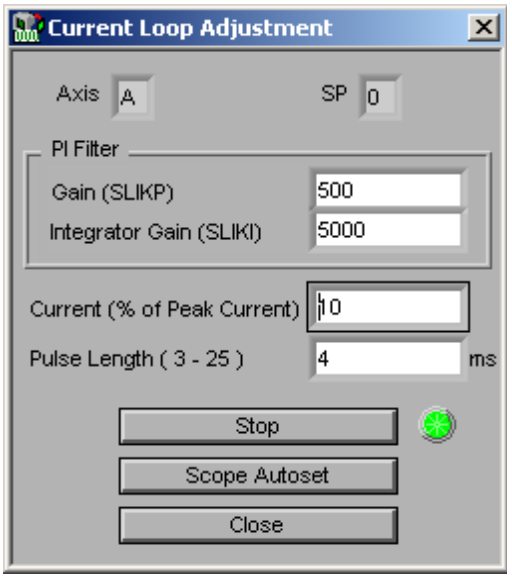


Figure 22 Current Loop Adjustment Dialog

2. For **Integrator Gain (SLIKI)**, enter **0**.
3. For **Gain (SLIKP)**, enter **100**.
4. For **Current (% of Peak Current)**, enter **10**.
5. For **Pulse Length**, enter **4** msec.
6. Click **Scope Autoset** to open the **SPiiScope**.

7. Increase the **Proportional Gain (SLIKP)** until the response waveform approximates a square and a small overshoot appears.
8. Increase the **Integrator Gain (SLIKI)** by hundreds, until you get a narrow overshoot of 10% to 20%.
9. Continue steps 7 and 8 till the motor response is satisfactory. Typical current loop adjustment results are shown in **Figure 23**.

Note	*During current loop adjustment, a relatively strong noise may emanate from the Control Module. This sound is normal.*

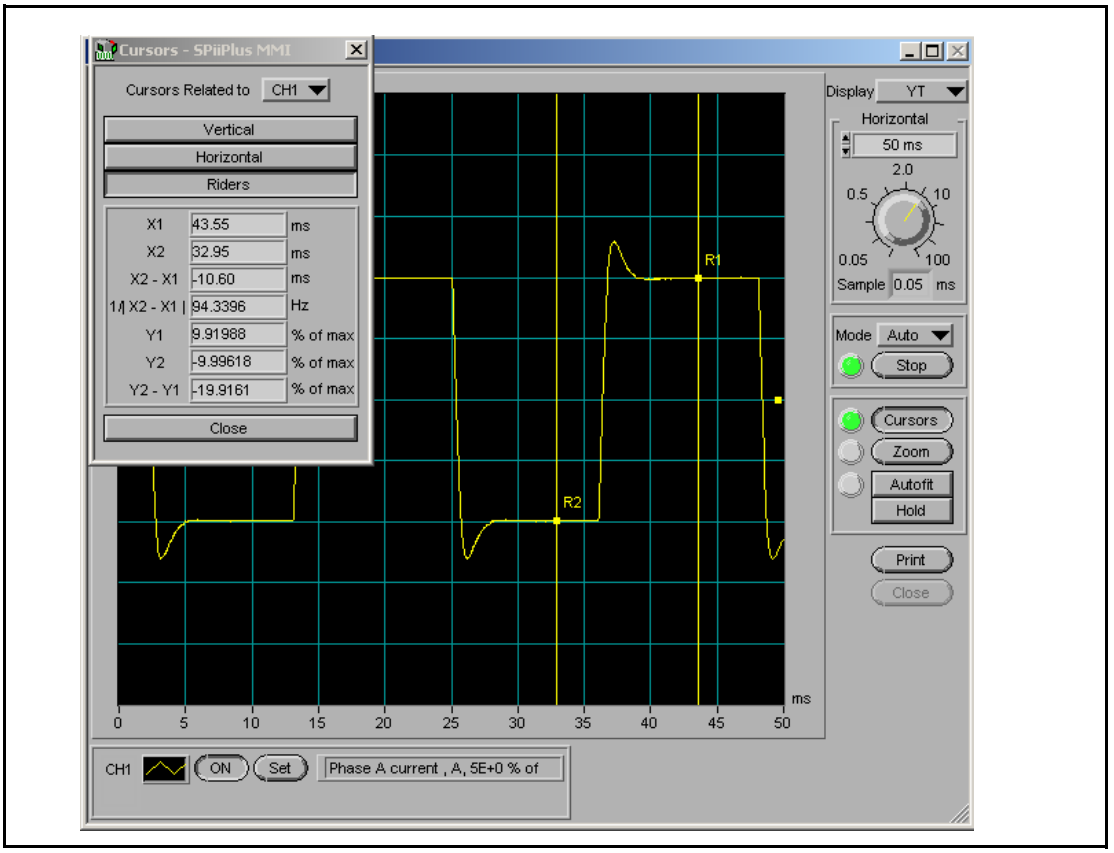


Figure 23 Typical Current Loop Adjustment Results

7 Commutation for DC Brushless (AC Servo) Motors

7.1 Commutation

7.1.1 Overview

To understand how the controller provides software commutation for DC brushless (AC Servo) motors, it helps to first look at hardware commutation in DC brush motors.

In a DC brush motor, a fixed magnetic field created by permanent magnets in the stator interacts with the armature current flowing in the rotor winding. The interaction of the current-carrying conductors with the magnetic field produces a torque (or force in the case of a linear motor). This torque/force is at its maximum value when the magnetic field vector is perpendicular to the resultant current vector.

The mechanical commutator of a DC brush motor distributes the motor current among the windings such that the resultant current vector remains perpendicular to the magnetic field vector at any position and speed. This process is referred to as commutation.

In a DC brushless/AC servo motor, an electronic drive takes the place of the mechanical commutator, keeping the resultant current vector perpendicular to the magnetic field vector by controlling the phase currents.

There are two types of drives for 3-phase DC brushless/AC servo motors:

1. **Input Drive**. This drive receives a single current command from the controller (10V ptp), reflecting the desired current amplitude, and takes care of the commutation by itself. From the controller point of view, this case does not differ from that of a DC brush motor – so there is no controller (software) commutation.
2. **Input Drive**. The drive receives two current commands from the controller, reflecting desired current for two of the motor phases. In this case the controller is responsible for commutation.

The desired current for two of the motor phases will be:

$I_A = I * \cos(\varphi)$

$I_B = I * \cos(\varphi + 120^o)$

where **I** is the amplitude of the current command and is the commutation angle (also referred to as commutation phase) measured in electrical degrees.

The third phase current is dependent on the first two phase currents:

$I_C = -(I_A + I_B)$

The three-phase currents generate a resultant current vector.

The commutation angle is dependent on the motor position:

$\varphi = \mathbf{k} * \mathbf{CP} + \theta$

where **CP** is the motor position, **k** is a conversion factor between position units and electrical degrees, and θ is an offset.

The **k** factor is a constant that depends on the (rotary) motor's number of poles (or linear motor's magnetic pitch) and the encoder resolution. The **SPiiPlus MMI Adjuster** calculates **k** according to the motor and encoder variables that you specified in **Chapter 5 - "Axis Configuration and Setup"**).

The θ offset keeps the resultant current vector perpendicular to the magnetic field. This offset is determined by the controller during the commutation setup process. During this process a current vector is generated, according to an arbitrary commutation phase ϕ_0. A position $\mathbf{CP_0}$ where the magnetic field of the motor aligns with this current vector is called a *detent point*. The offset θ can be calculated at the detent point as:

$\theta = \phi_0 - K * \mathbf{CP_0} + \mathbf{90°}$

Once this relation is known, the current vector can be kept perpendicular to the magnetic field, thereby achieving maximum motor performance (maximum torque/current ratio).

Since the commutation angle,ϕ, depends on the motor position, the offset has to be determined only once for an absolute encoder but after every power-up for an incremental encoder – since the motor position is not known.

Therefore, when working with an incremental encoder and a DC brushless motor, commutation setup must be executed after every powerup.

7.1.2 Commutation Terminology

Commutation Adjustment – Commutation process carried out with the **SPiiPlus MMI Adjuster** as part of setting up the controller.

Commutation Startup Program – ACSPL+ program that can be generated automatically during the **Commutation Adjustment.** The program is used to retrieve commutation. It can be executed after every powerup (once the system has been setup). The startup program can be based either on bringing the motor to a detent point (usually involves movement) or on **automatic commutation (commut** command**),** which is fast commutation retrieval using a closed-loop algorithm (and involving almost no motor movement).

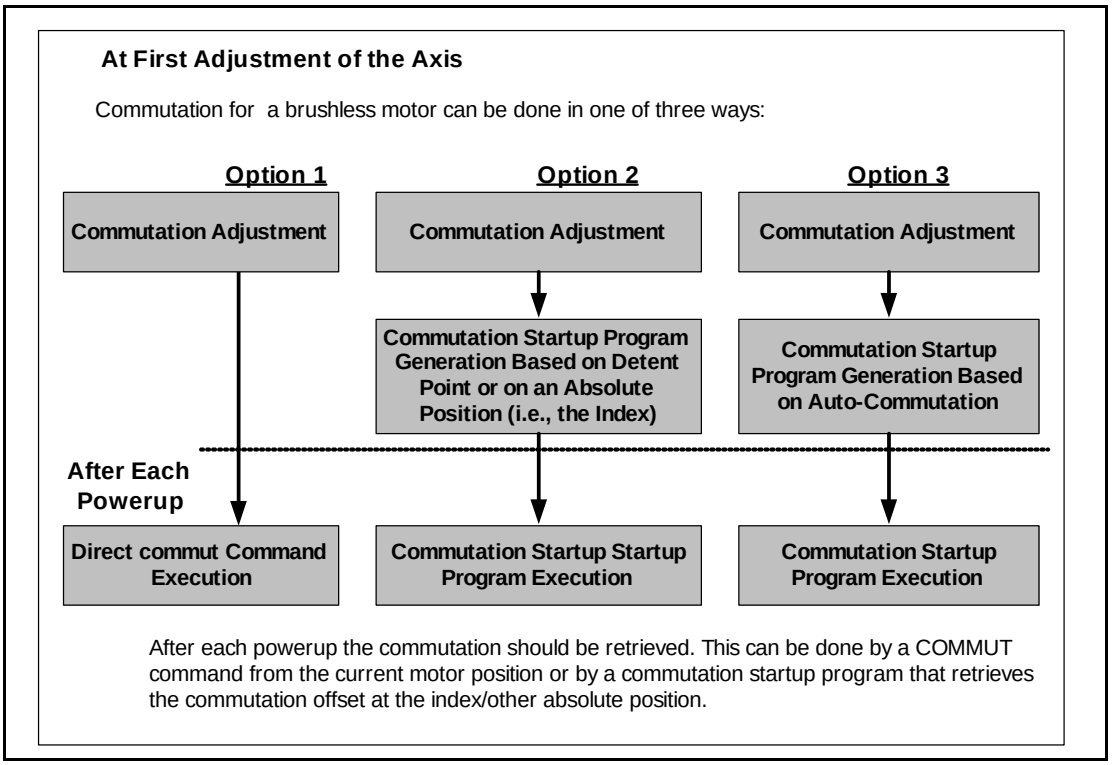


Figure 24 Commutation Options

The **SPiiPlus MMI Adjuster** is used to perform Commutation Adjustment and, optionally, to generate a startup program.

7.2 Commutation Adjustment

Commutation Adjustment is required at the first system startup.

The **SPiiPlus MMI Adjuster** executes a program that performs Commutation Adjustment setup. The program supports several commutation schemes (procedures). You select the most appropriate scheme for your application. In most cases, the commutation scheme will include the following steps:

1. Find a detent point.
2. Identify the phase sequence.
3. (Optional): Verify variables (number of poles or magnetic pitch, encoder resolution) by moving the motor several magnetic pitches and comparing the encoder feedback with the expected results.
4. (Optional): Move to the index and save the commutation phase at that point to the controller flash memory.

7.2.1 Commutation Adjustment Procedure

Model	
	*SPiiPlus CM: Current loop adjustment must be done before starting Commutation Adjustment Setup.*

Note	
	*In both Commutation Adjustment and Commutation Startup programs that are NOT based on **automatic commutation** the motor is moved by moving the current vector, which pulls the rotor with it.*

Warning	
	***When the current vector is aligned with the magnetic field during commutation, the motor can jump. The maximum jump is one magnetic pitch (180 electrical degrees) of the motor in either direction. If the motor bumps into an obstacle, the commutation setup algorithm will attempt recovery. The recovery may involve additional abrupt moves.*** ***To prevent possible damage or injury, it is recommended that the motor be initially positioned at least one magnetic pitch away from any obstacles.***

1. Open **MMI → Setup → Adjuster → Commutation**.

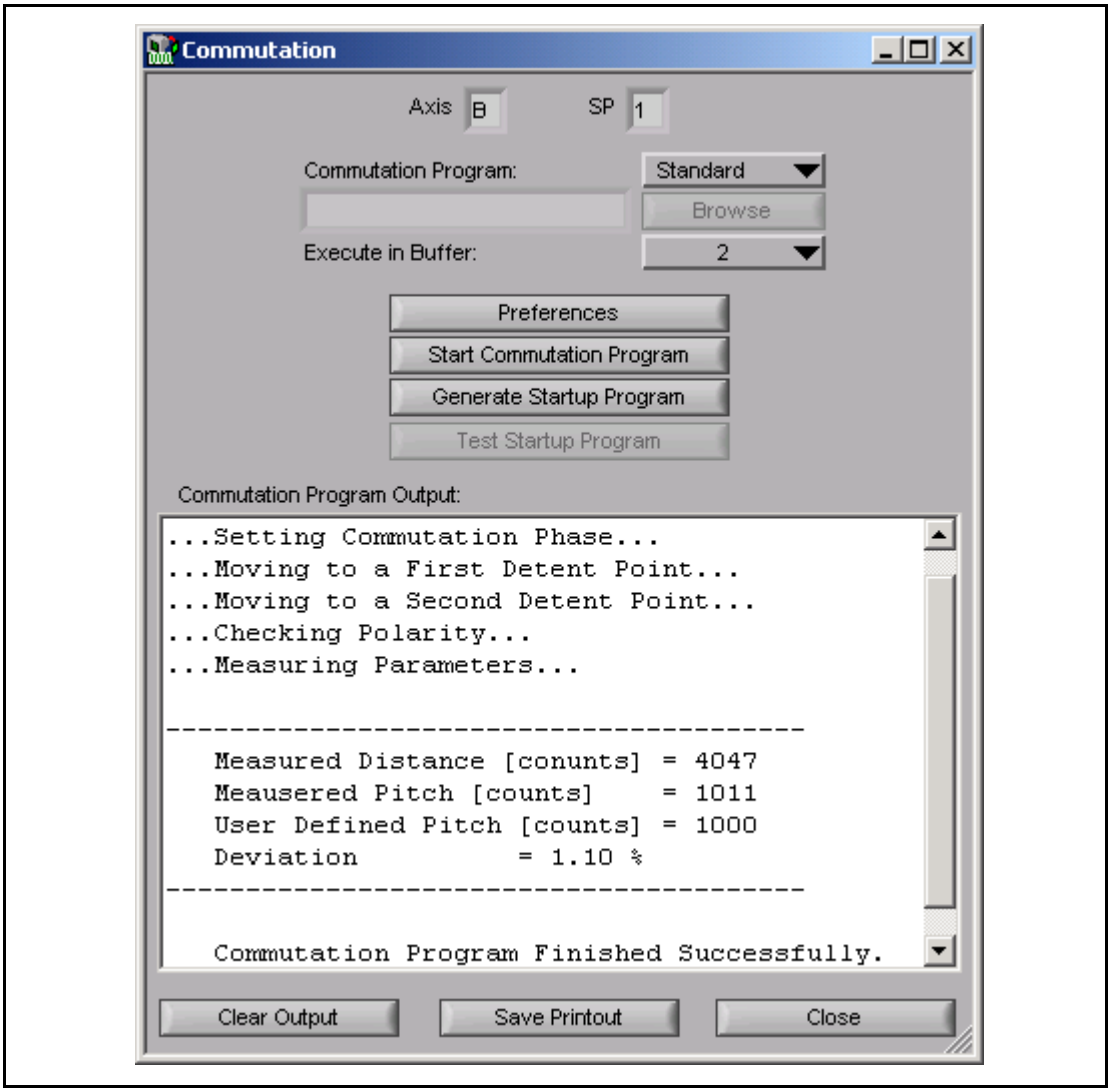


Figure 25 Commutation Dialog

2. Select one of the two **Commutation Program** options:
   - **Standard**: Starts with built-in default commutation scheme.
   - **User Defined**: Enables **Browse** button for selecting a commutation program that was saved in a file (in this case the user needs to prepare his own Commutation Startup program.
3. Select the desired **Buffer** for the commutation execution. (The controller supports up to 9 simultaneously executing program buffers.) Usually, you will want to select an empty buffer to avoid overwriting an existing program.
4. Click **Preferences** to open the **Preferences** dialog box. The **Preferences** dialog box defines the commutation scheme and variables for the commutation process.

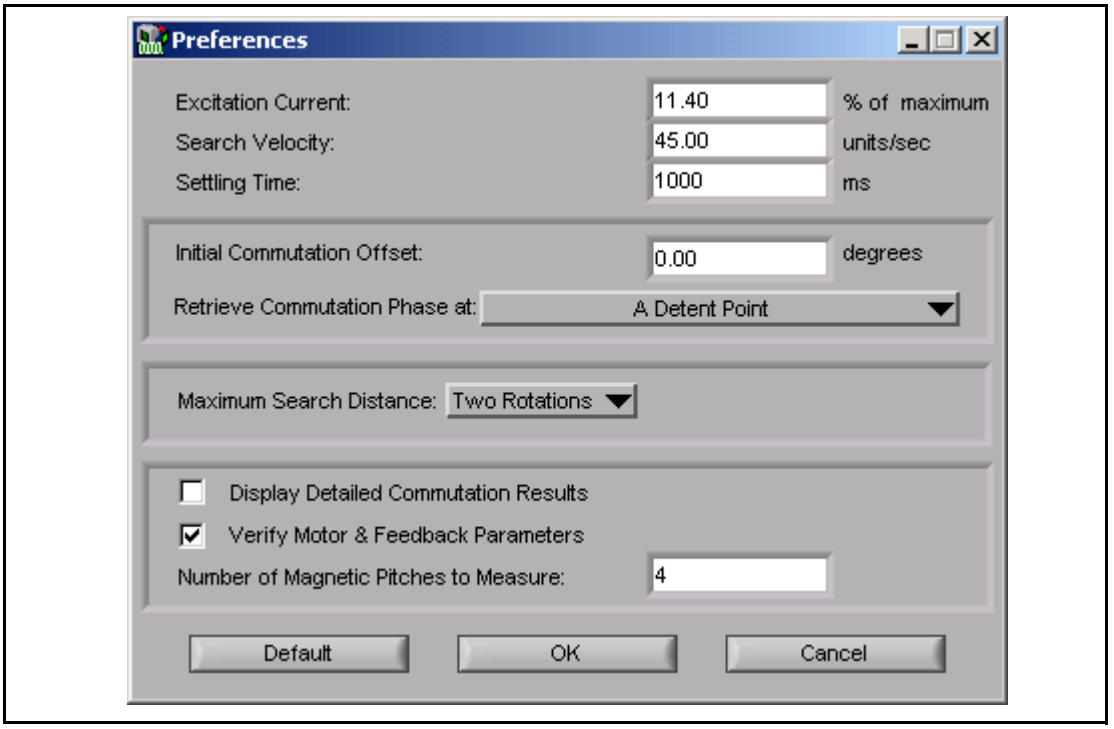


Figure 26 Preferences Dialog

5. Enter an **Excitation Current**. This determines the amplitude of the current vector that is used for commutation setup. The value should not exceed the nominal current ratings of the drive and the motor. The default value is: 0.95 x **XRMS** - see **Section 5.3.6.1 - "Setting Axis Maximum Current/Torque Limits"**.

Note	
	*The following considerations must be taken into account:* • *Due to friction and load, a low excitation current may result in poor alignment between the current vector and the magnetic field, resulting in unsatisfactory commutation. It is recommended to set the excitation current high enough to generate torque/force that is at least four times higher than the friction and load. For a rotary motor,* $\text{Torque}_{max} = K_T \times I_{excitation}$. • *On the other hand, too high an excitation current with a low friction system (inadequate damping) may result in excessive oscillation and even in mechanical damage.*

6. Enter a **Search Velocity**. This determines the velocity of the current vector during the commutation process. During this process the motor position follows the current vector in order to align the magnetic axes. It is recommended to set a velocity lower than one magnetic pitch per second and higher than one fifth of a magnetic pitch per second.
7. Enter a **Settling Time**. This determines the time assigned to the motor for settling at detent points.

8. Enter an **Commutation Adjustment Offset**. This determines the initial value of the commutation phase. It therefore determines the initial orientation of the current vector and the initial detent point. By properly setting this offset you can avoid initial jump in a system with a predefined startup position. For other systems, the offset can be used to determine a specific target position for the initial jump.
9. Select a **Commutation Scheme**. In all the schemes except the powerup position scheme, the following steps are performed:

   a) Settle at a first detent point.

   b) Move the current vector to identify the phase sequence. Correct for wrong polarity automatically.

   c) If the feedback indicates that the motor has moved less than 50% of the expected distance, it is assumed to have run into an obstacle and the program tries to recover in the opposite direction.

   d) Retrieve the commutation phase according to the selected scheme. Considerations when selecting a **Commutation Scheme**:

   - Based on detent point (first scheme): If the feedback device has no index, this is the only scheme that can be used. If the feedback device does have an index it is recommended to use one of the "index" schemes so that the index can be used as an absolute reference point for commutation phase retrieval at subsequent powerups.
   - Based on index (six schemes): Saves the value of the commutation phase at the index position. This value can be used at subsequent system powerups to achieve the same commutation result. Six variations of this scheme are available:
     - First index in positive direction
     - First index in negative direction
     - First index next to right limit switch
     - First index next to left limit switch
     - First index next to right hard stop (if the system is not equipped with limit switches)
     - First index next to left hard stop (if the system is not equipped with limit switches)

   e) Based on known powerup position: Useful when an axis always starts from a known position. For example, a vertical axis.
10. Select or enter a **Maximum Search Distance**. This sets the maximum allowed distance for searching for targets (index or limits). It is recommended to set the variable according to the operational distance.
11. Select **Display Detailed Commutation Results** to receive commutation phase data (in electrical degrees).
12. Select **Verify Motor & Feedback Parameters** to have the system verify the encoder and motor variables during the Commutation Adjustment process. If the variables seem to be configured wrong, a general message will be displayed stating that improperly configured variables are a possible reason for failure. Once corrected you will be able to continue. In a

case of more than 20% deviation between the defined and measured pitch, the commutation process reports failure.

Caution	
	***Deviation between defined and measured pitch does not necessarily indicate poor commutation quality. For example, incorrect definitions of feedback variables may be compensated for by high friction, thus the deviation may be low but the commutation will be poor.***

13. Click **OK** and in the **Commutation** dialog box, click **Start Commutation Program.** If the commutation process finishes successfully, bit 9 (Commutation OK) of **MFLAGS** is set.

Note	
	*If changes are made to the motor, encoder, or their wiring, then full adjustment procedure including Commutation Adjustment must be repeated.*

7.2.2 Automatic Commutation

Caution	
	***Automatic Commutation will perform well only if the axis has been configured and the position and velocity loops have been adjusted.***

Warning	
	***If any hardware change is made to the motor, encoder, or their wiring, for example, reconnection of motor or encoder phases, Commutation Adjustment without automatic commutation must be executed again.***

Automatic commutation is a way to retrieve the commutation offset automatically, using closed-loop control.

Commutation setup (without using automatic commutation) is required once: during Commutation Adjustment. Once this has been done, automatic commutation is recommended at every controller powerup.

Advantages of automatic commutation:

- Minimizes initial jump.
- Allows closed-loop motion already at startup.
- Minimizes problems with vertical axes.

- Minimizes problems when the axis is in proximity to a hard-stop.
- Minimizes problems when the working area is smaller than the electrical cycle.
- Allows fast commutation retrieval.

After completing the initial system setup and adjustment, there are two options for executing automatic commutation in subsequent powerups:

Use the ACSPL+ command: **COMMUT <axis>** (see the *SPiiPlus ACSPL+ Programmer's Guide*). Use a Startup Program generated with Automatic Commutation selected.

7.3 Commutation Startup Program

A commutation startup program is a simple ACSPL+ program (program that runs on the controller) that retrieves commutation. Typically, you will want to make the startup program (together with a homing program) part of the controller's AUTOEXEC routine, which ensures that the startup program is run after every controller powerup.

Note	
	*Commutation adjustment must be carried out after the very first controller startup (and after any change to motor, encoder, or their wiring). The startup program is used after all subsequent powerups.*

The startup program follows the same scheme that was used for the Commutation Adjustment but with only the following steps:

1. Find a detent point.
2. (If the commutation phase was saved in Commutation Adjustment): Move to the index and read the commutation phase for that point from the controller flash memory.

Normally the phase sequence and motor and encoder variables do not change after Commutation Adjustment so they are not addressed by the startup program.

Usually the startup programs are based on bringing the motor to a detent point (which involves a movement of the motor. Nevertheless, there is an option to generate start up programs based on **Automatic Commutation**, see **Section 7.2.2 - "Automatic Commutation"**. This option affects the execution of the **Commutation Schemes** as follows:

- **Detent point** scheme: Movement of the axis is minimal.
- **Index** scheme: The axis moves to the index in closed-loop (commutated) motion.
- **Powerup position** scheme: not affected.

7.3.0.1 Generating a Commutation Startup Program

Warning	
	***When the current vector is aligned with the magnetic field during commutation, the motor can jump. The maximum jump is one magnetic pitch (180 electrical degrees) of the motor in either direction. If the motor bumps into an obstacle, the commutation setup algorithm will attempt recovery. The recovery may involve additional abrupt moves.*** ***To prevent possible damage or injury, it is recommended that the motor be initially positioned at least one magnetic pitch away from any obstacles.***

1. Open **MMI → Setup → Adjuster → Commutation**.
2. If necessary, modify preferences.

3. Select **Generate Startup Program** to generate the startup program.

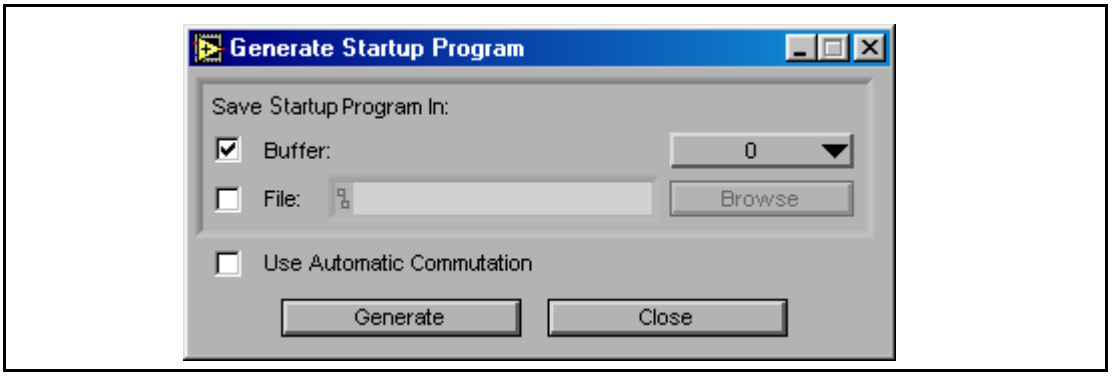


Figure 27 Generate Startup Program Dialog

4. Select where to save the startup program.
5. Select **Automatic Commutation** if desired (recommended).
6. Click **Generate**.
7. If this is the initial axis setup, continue to **Open Loop, Index Verification,** and **Axis Adjustment**. After those steps, you can return and perform automatic commutation, if required.

7.4 Commutation Startup Program Test

1. Open **MMI → Setup → Adjuster → Commutation**.
2. Click on **Test Startup Program** to run the startup program.
   Progress messages are displayed in the **Commutation Program Output** area of the window.

Note

*In single loop control, commutation retrieval is based on the primary index (***IND** *variable), while in dual loop control it is based on the secondary index (***E2IND***).*

Advanced

*The startup program includes a local variable (***SP_Fail_Code***) that stores a termination code in the event that the program fails. You can check* **SP_Fail_Code** *to investigate the cause of a failure.*

To check the code, open **MMI → Terminal** *and enter the command?<`buffer_number`>: `SP_Fail_Code`. Then look the code up in the following table.*

For example, assuming that the startup program is in buffer 1, enter the command:`?1: SP_Fail_Code`.

Termination Status	SP_Fail_Code
Program terminated successfully.	0
Motor error. (check **MERR** variable).	1
Index not found within the search range.	2
Limit switch (where limit switches are used for the commutation scheme) or hard stop (where hard stops are used for the commutation scheme) not found within the search range.	3
Unable to move away from the limit switch (where limit switches are used for the commutation scheme).	4

7.5 Troubleshooting Commutation

If the motor does not move during **Commutation Adjustment** or **Commutation Startup** program execution, consider the following:

- Check the connections between the controller and the drive and motor.
- If a high friction or active load is applied, increase the **Excitation Current** and try again.

If **Commutation Adjustment** or **Commutation Startup** program execution displays a message about wrong motor variables, return to the **MMI → Axis Setup** window and verify the following:

- **Number of Poles** or **Magnetic Pitch** are specified correctly.
- Encoder variables are specified correctly.

If the motion failed during Commutation Adjustment or Commutation Startup program execution, probable causes are:

- The required **Excitation Current** is greater than the **XRMS** value. If the motor and drive rating allow it, try to increase the **XRMS** value.
- There is an obstacle in the path of the motion.

Another problem that can arise is the case where **Check Motor & Feedback Parameters** is not selected in the **Preferences** window. Inaccurate variables can cause the commutation process to report successful completion even though the commutation is actually wrong. Therefore it is recommended to select this field.

- Possible reasons for failure of the commutation process:
- Hard stop: motor has bumped into a hard stop or an obstacle and is unable to move.
- Excitation current too low: results in poor field alignment.
- Inaccurate motor and feedback variables.
- Hardware problems, such as encoder or drive fault, wiring error, or bad grounding.

8 Open Loop and Index Verification

Note	
	*This chapter applies only for servo motors (DC brush or AC servo/DC brushless).*

Open loop verification should be performed to verify proper operation of the following:

- Feedback device count is in the expected direction and is correlated with the drive command
- Index signal is latched.
- Drive and motor hardware is connected.

8.1 Feedback, Drive Command and Index Verification

1. Open **MMI → Setup menu → Adjuster → Open Loop Verification**

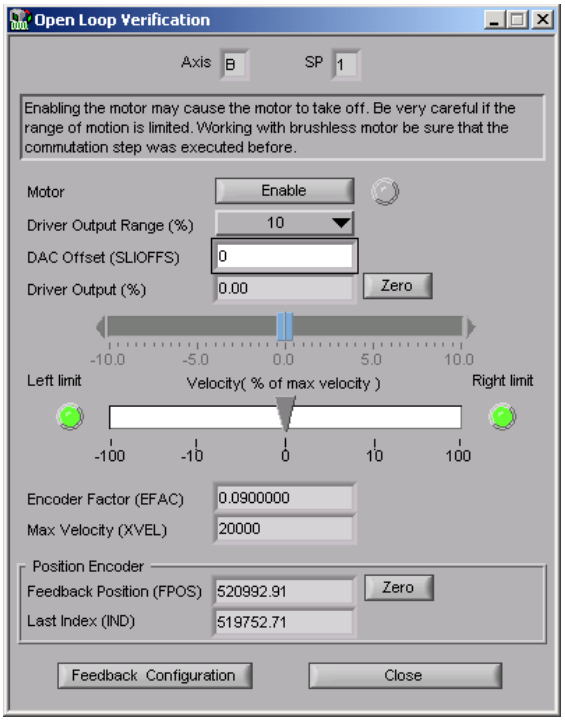


Figure 28 Open Loop Verification Dialog

Warning	
	***If the motor is connected, enabling the drive may cause a motor jump. Verify that the axis travel is clear of obstacles.*** ***Be ready to engage the Emergency Stop switch.***

2. Click **Enable** to enable the drive. Verify that the indicator turns green and the button changes to **Disable** (which is now the operation that it triggers).

3. Click **Zero** to reset the encoder position.
4. Enter **10** for the **Drive Output Range**. This determines that the drive command will be 10% of the maximum. This means that the maximum drive command voltage will be +/-1V.
5. Drag the **Drive Output** slider slowly to the right until the motor moves. If the motor doesn't move (due to friction), increase the **Drive Output Range** (%) (with the slider or by entering a value) until the motor moves.

   Verify that the velocity gauge indicator moves to the right (same direction as you moved the slider). If the velocity arrow moves in the opposite direction, click **Feedback Configuration** and select **Invert Drive Output**.

Warning	
	***If the direction of the velocity gauge is opposite to that of the Drive Output slider, it is an indication of positive feedback. This is likely to cause a motor run away.***

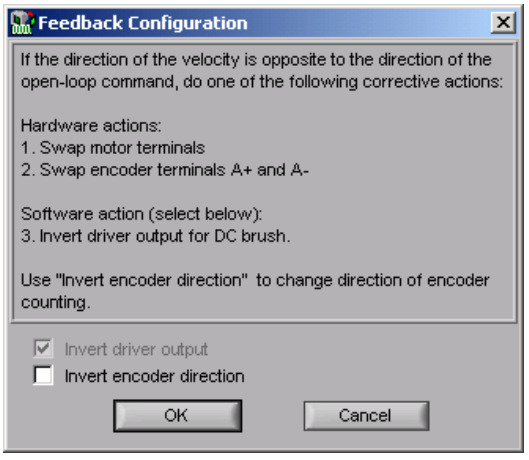


Figure 29 Feedback Configuration Dialog

6. Verify that the **Feedback Position** increases positively when the motor direction is positive (in the context of the end-application) – otherwise click **Feedback Configuration** and select **Invert Encoder Direction**.
7. Verify that the **Last Index** field is updated every time the motion passes the encoder index. In a rotary motor with one index, the difference between two consecutive index readings should equal the number of counts per revolution (assuming that the user units are counts).
8. For DC brush motor only: If there is bias (an offset) in the drive (motor moves while the controller command is zero), then with **Driver Output Range** (%) set to zero, use the **DAC Offset** (**SLIOFFFS)** field to compensate.

9 Position and Velocity Loops

9.1 Overview

SPiiPlus MMI provides a window with a soft oscilloscope for adjusting the position loop and velocity loop. The **Position** & **Velocity Loops** window includes tabs for variables that are specific to dedicated features (for example, notch filter) or special capabilities (for example, support for Nanomotion piezo ceramic motors). Since adjustment of the position and velocity loops is dependant on the **EFAC** and **XVEL** variables, these variable values are displayed for reference (read-only). **EFAC** can be modified from the **Axis Setup** window (see **Section 5.2 - "User Units"**) and **XVEL** from the **Safety Parameters** window (see **Section 5.3.5 - "Maximum Velocity And Acceleration"**).

Caution	
	***Trying to adjust the position and velocity loops when the* XVEL *or* EFAC *variables are not set right will produce poor results. Verify that these variables are defined correctly (to fit your application/ requirements) before starting to adjust the loops.***

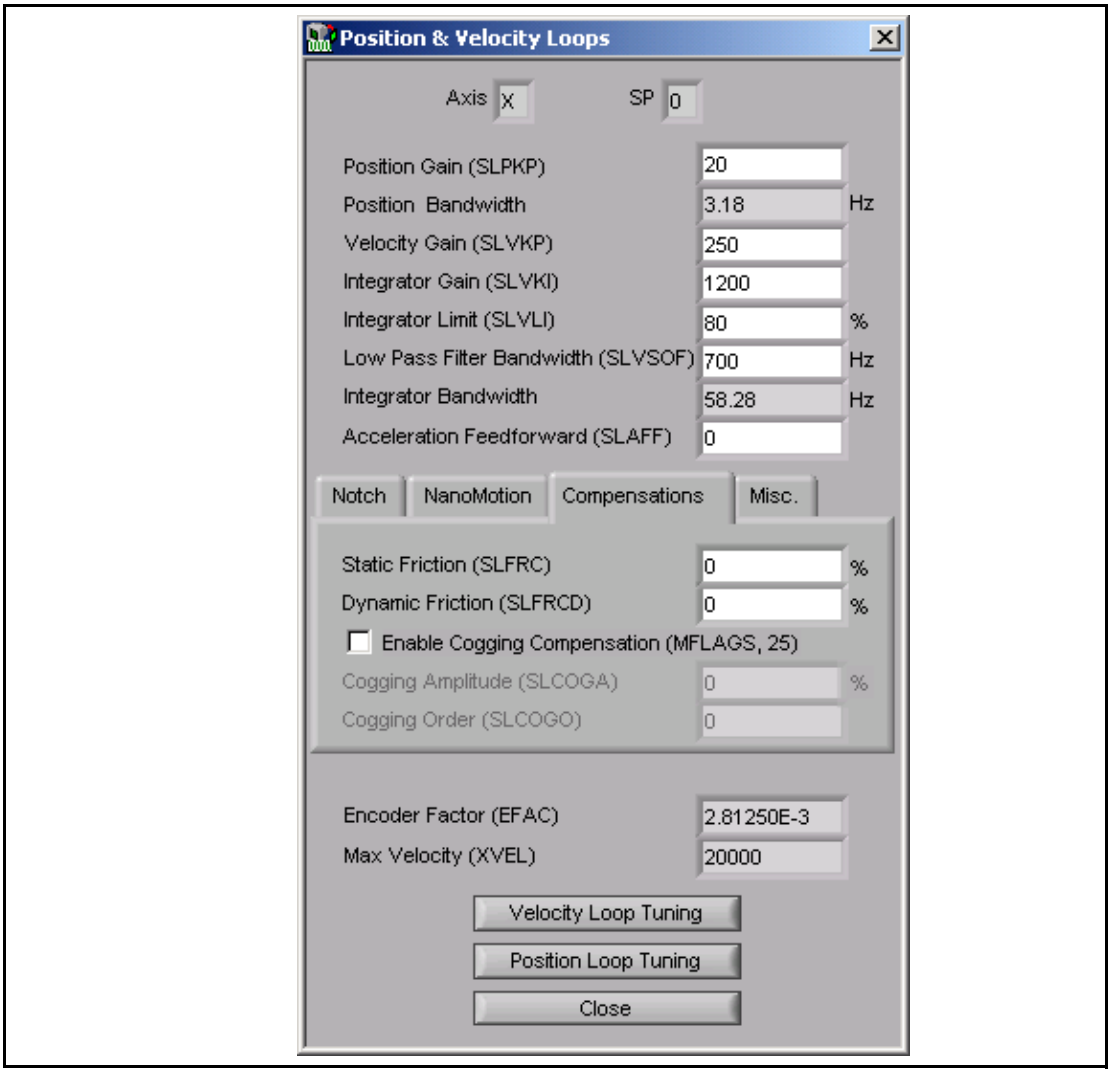


Figure 30 Position and Velocity Loops Dialog

Note	*The PIV control algorithm enables you to adjust each loop separately.*

9.2 Adjusting the Velocity Loop

1. Open **MMI →Setup → Adjuster → Position & Velocity Loops**. The **Position & Velocity Loops** window and the **Scope** window open.
2. Click **Velocity Loop Tuning**. The **Position & Velocity Loops** window expands with velocity loop variables and commands.

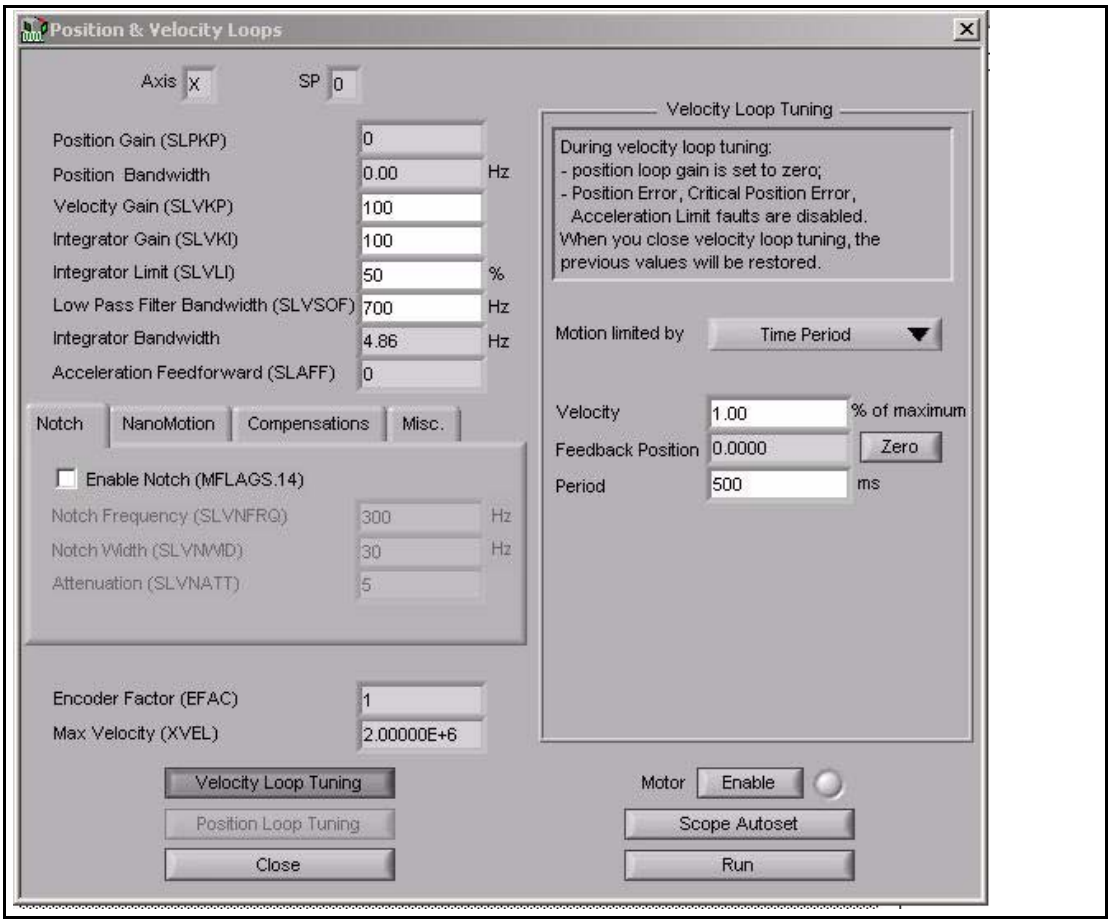


Figure 31 Position & Velocity Loop Dialog

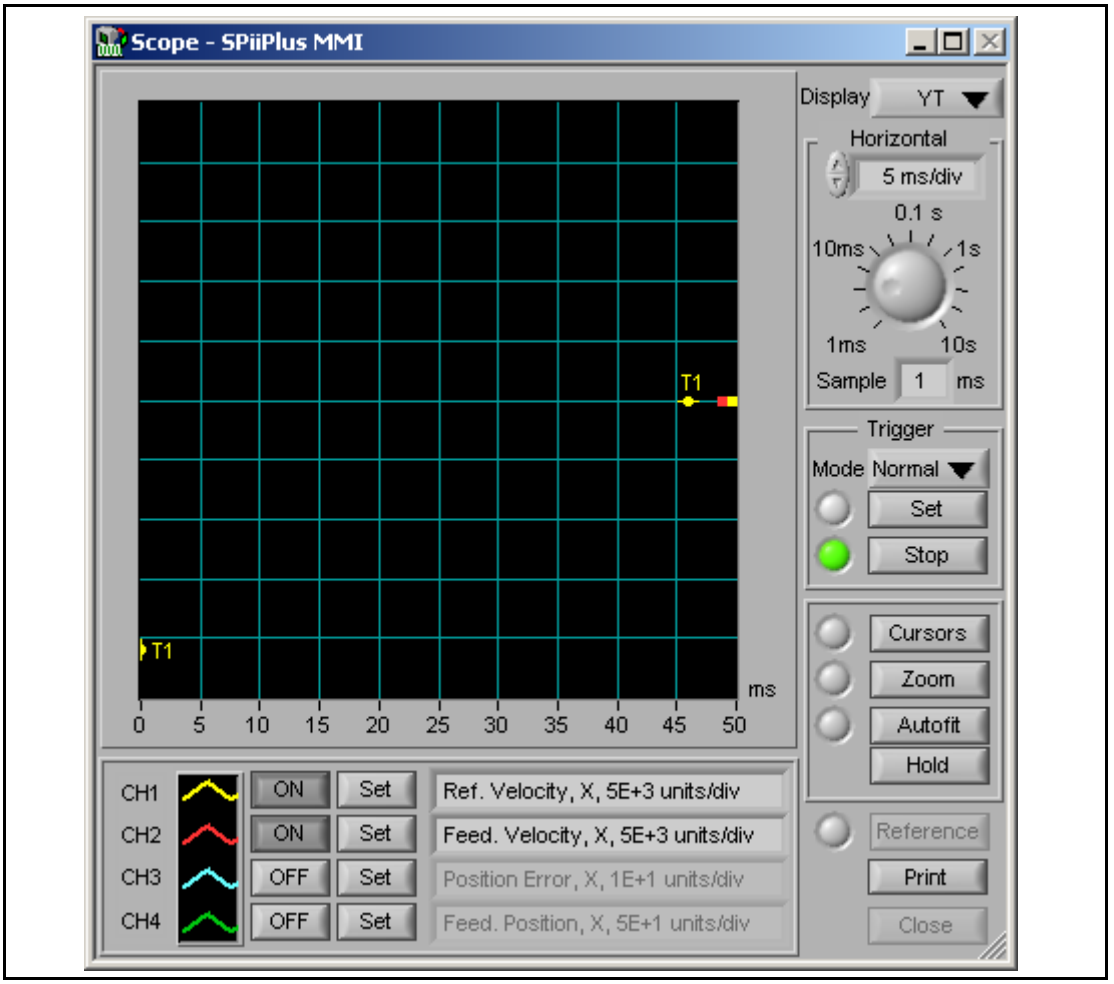


Figure 32

3. In the **Motion Limited By** field, select a limiting factor described in **Table 17**. The selection determines which other **Velocity Loop Tuning** fields are displayed.

Table 17 Limiting Factors for Velocity Loop Tuning

Limit Factor	Description
Time Period	Appropriate for a rotary system. In the **Period** field enter the time in ms to move in each direction.
Positive Direction Only	Allows the motor to move in the positive direction only.
Negative Direction Only	Allows the motor to move in the negative direction only.
Between Two Points	Appropriate for a linear system. To zero the feedback position, click **Zero**. To read the feedback position and assign it to a **Point**, click the corresponding **Read** button.

4. For **Low Pass Filter Bandwidth,** enter **650Hz**.
5. For **Integrator Gain**, enter **0**.
6. For **Velocity Gain**, enter **100**.
7. For **Integrator Limit**, enter **50%**.
8. Verify that the **Velocity** field is set to **10**.
9. Click **Enable** to enable the drive.
10. Click **Run**. Reciprocated motion begins.
11. Click **Scope Autoset** to select the reference velocity and feedback velocity as input for the **Scope** window.

    For the next two steps, refer to the motion profile displayed in the **Scope** window.

12. In the **Position & Velocity Loop** window, double the **Velocity Gain** until the profile approximates a square with a small overshoot.
13. Increase the **Integrator Gain** by hundreds, until you get a narrow overshoot of 10-20% in the waveform.

Note	
	*It is recommended that the **Integrator Bandwidth** not exceed 50Hz.* *(Integrator Gain)/20 = Integrator Bandwidth*

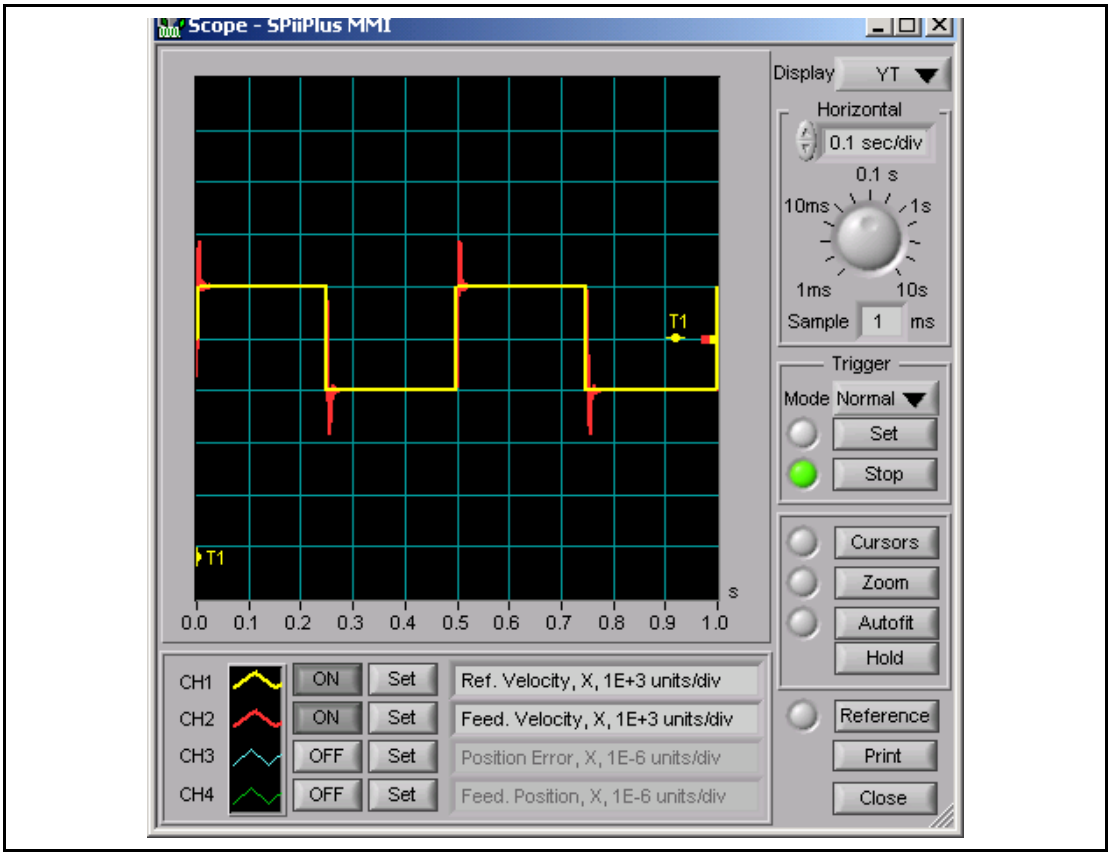


Figure 33 Example of Good Velocity Loop Tuning

14. Continue adjusting the **Velocity Gain** and **Integrator Gain** until a satisfactory profile is achieved, as illustrated in **Figure 33**.

9.3 Adjusting the Position Loop

Before you start to adjust the position loop, configure the following parameters

- **First point** and **Second point** – to define the travel of the axis during position loop adjustment.
- **Velocity**, **Acceleration**, **Deceleration**, and **Jerk –** to define motion profile **variable**s. The definitions apply only for position loop adjustment using this window. They are separate from the motion profile definitions used by the controller during normal operation.
- **Dwell** – time in milliseconds to wait in each direction.

1. Select **Overcurrent** to enable detection of that fault.
2. To zero the **Feedback Position**, click **Zero**. To read the feedback position and assign it to a **Point**, click the **Read** button next to the **Point**.
3. **S**et the distance between the first and second **Point** to be large enough that the motor will reach a constant velocity while remaining within the system limits.
4. Select the desired **variable**s of trial motion. Set the **Velocity**, **Acceleration** and **Deceleration** to 50% of the maximum required for the motion control application.
5. Click **Run**. Reciprocated motion starts.
6. Click **Scope Autoset**. This allocates Scope channels to the position loop **variable**s: **RVEL**, **FVEL**, and **PE**.
7. Increase the **Position Gain** to minimize position error during acceleration and deceleration.
8. Increase the **Acceleration Feedforward** to minimize position error during acceleration and deceleration.

Note	
	*The theoretical Acceleration Feedforward can be calculated as:* *ACC_FF=[2^16 x 10^7 x EFAC / ACC] x.0.1*

9. To improve the adjusting results, you may change the **Velocity Gain**, **Integrator Gain** or **Low Pass Filter Bandwidth.** You can also activate a notch filter, see **Section 9.4.2 - "Using the Notch Filter"**. A typical position loop profile is shown below.

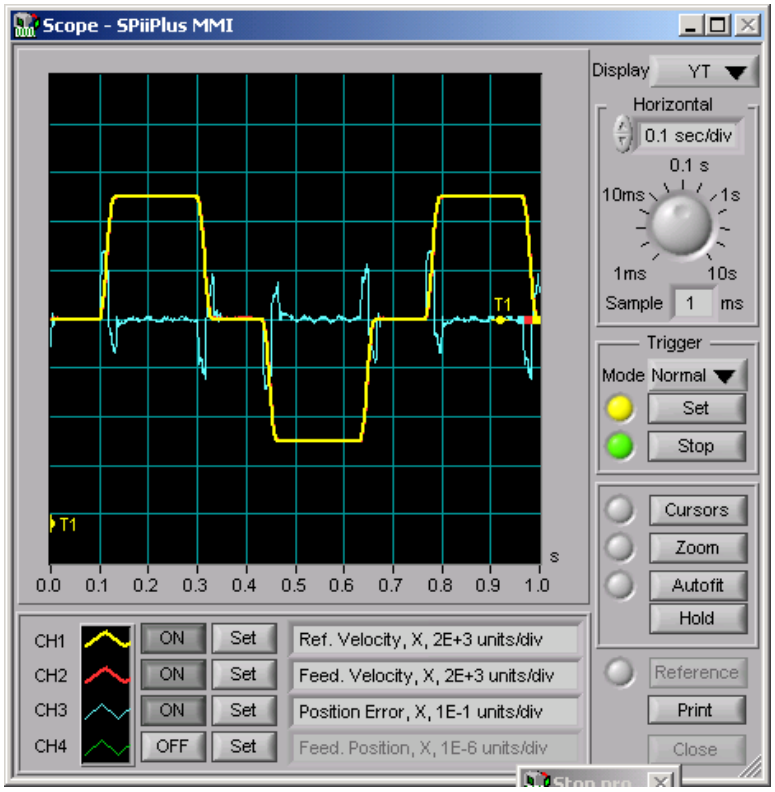


Figure 34 Typical Position Loop

Advanced	*The* **Scope Autoset** *button selects the* **RPOS**, **FPOS** *and* **PE** *variables of the axis. Advanced users can also monitor the* **DOUT** *variable in channel #4. It will provide an indication about the command to the drive at any stage of the motion. The* **DOUT** *full range is between (-32,768) to (32,767) which is translated in the range of 10V ptp analog output command to the drive.*

9.4 Advanced Position & Velocity Loop Variables

Advanced 	*This topic is for users who are familiar with SPiiPlus ACSPL+ programming.*

9.4.1 Friction Number (SLFRC) Implementation

The velocity friction variable **SLFRC** is a percentage (range: 0% to 50%) of the maximum output. The variable sets the initial value of the velocity-loop integrator when the motion-profile starts. Generally, **SLFRC** should be set to zero. For high friction load, increasing **SLFRC** shortens the start motion delay by compensating for the friction torque or force.

9.4.2 Using the Notch Filter

The notch filter is designed to attenuate mechanical resonance at a specific frequency.

Note 	*The motor must be disabled while the notch filter is enabled.*

1. Verify that the notch filter is disabled (on the **Notch** tag, the **Enable Notch** check box is cleared, meaning that **MFLAGS** bit 14 is zero).
2. Use a spectrum analyzer to measure the mechanical resonance frequency.
3. Select the **Notch Filter** check box and for the **Notch Frequency**, enter the frequency that you have measured.
4. Set the **Notch Width** to 5%-10% of the notch frequency and the attenuation to 5.
5. Run a motion profile, and watch the feedback velocity response on the Scope.
6. Optimize the oscillation by changing the width and the attenuation of the filter. Maximum allowed attenuation is 20. Maximum allowed width is 33% of the notch frequency.

10 Motion Defaults

10.1 Criteria for Determining Whether Motor is in Position

Two variables determine when the motor is considered to be in position:

1. **Settling Time** (**SETTLE**) - Amount of time (in milliseconds) that position error (**PE**) must stay within the Target Envelope (**TARGRAD**) range: **-TARGRAD** to +**TARGRAD.** Default: 0.
2. **Target Envelope (TARGARD) –** Maximum **PE** allowed during the **Settling Time** (**SETTLE**). **PE** is the difference between **RPOS** (motor reference position) and **FPOS** (motor current feedback position). Default: 1.

10.1.1 Defining Motor In-Position Criteria

1. Open **MMI → Setup → Configurator**

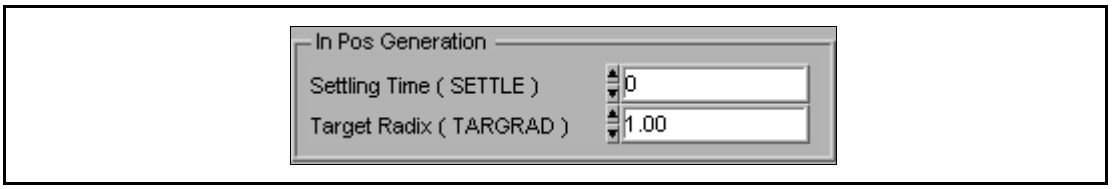


Figure 35 In Pos Generation Dialog

2. Enter values for **SETTLE** and **TARGRAD**.

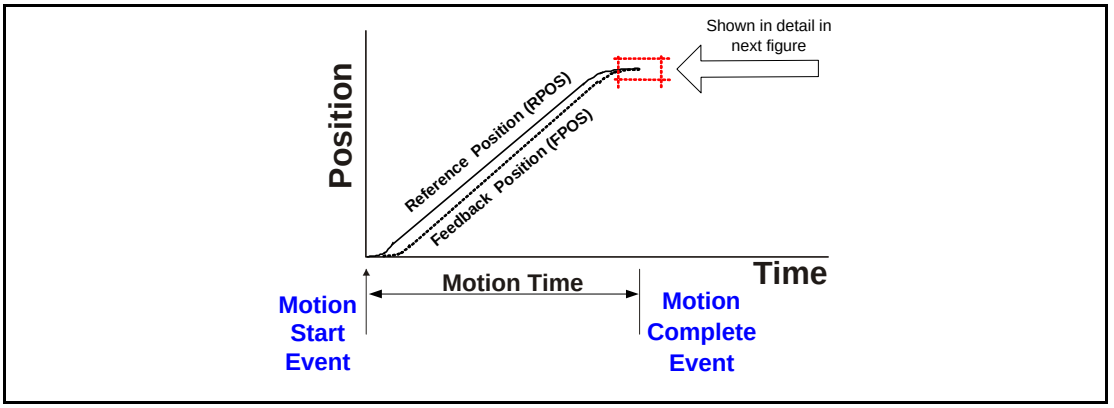


Figure 36 Motor Position Profile - General

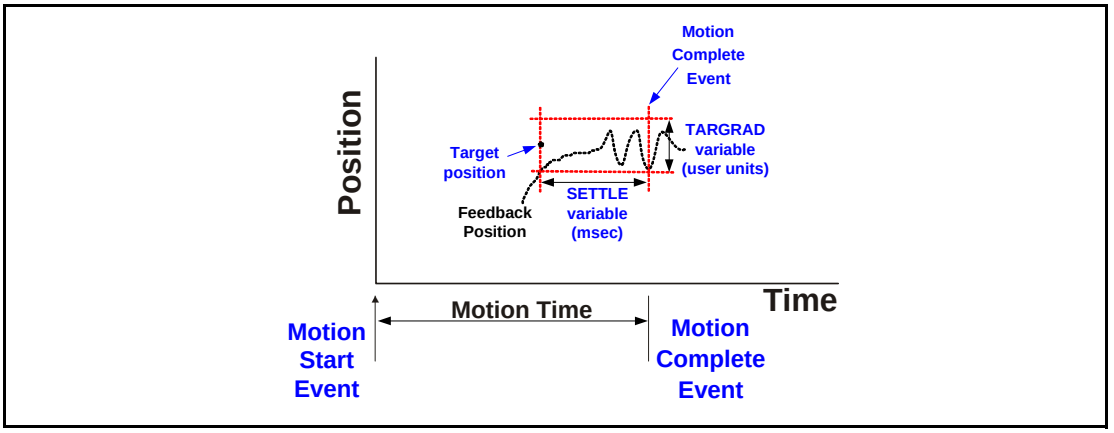


Figure 37 Motor Position Profile - Detail

10.2 Motion Profile

The following variables are maximum values used when the controller generates the axis's motion profile.

- **Velocity (VEL)**.
- **Acceleration (ACC)**.
- **Deceleration (DEC)**.
- **Jerk (JERK)**.
- **Kill Deceleration (KILL)** – Deceleration after the motion is killed by the user or fails due to a fault.
- **Minimum Velocity (NVEL)** – Normally the motion profile starts from zero velocity. **NVEL** is used to start the profile from a non-zero value (start with a "jump"). This variable is often used with stepper motors.
- **Feedback Velocity Filter** (**FVFIL**) – Filter applied to the feedback velocity (**FVEL**). The parameter does not affect servo loop - affects indication only. For nonzero **FVFIL**:
  $FVEL_n = (FPOS_n - FPOS_{n-1}) * K* (1 – FVFIL/100) + FVEL_{n-1}* FVFIL/100$
  where K is a scaling factor that reduces **FVEL** to user units per second.

10.2.1 Setting Motion Profile Variables

To set the motion profile variables, open **MMI → Setup → Configurator** and enter variable values.The Configurator dialog appears as in **Figure 38**.

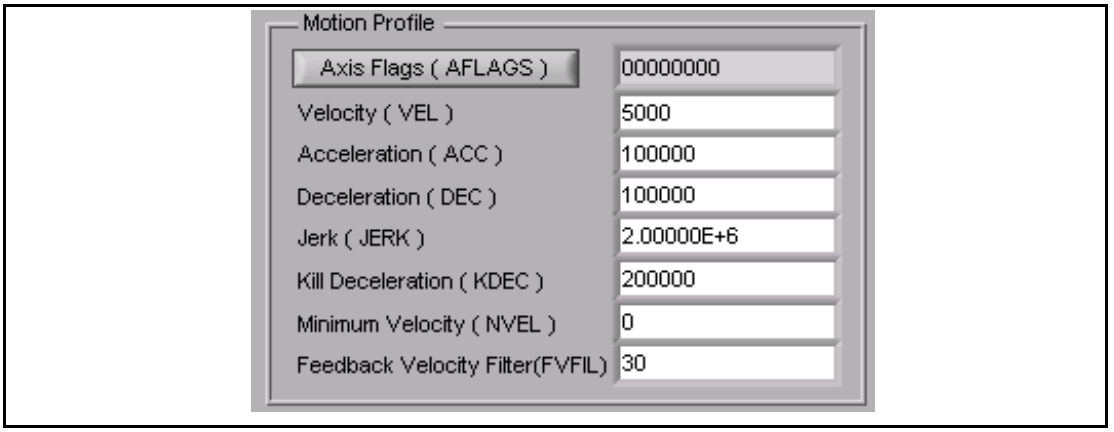


Figure 38 Configurator Dialog

11 Saving, Copying, and Protecting Axis Variables

The controller RAM is erased when the controller undergoes a hardware. A hardware reset can be user initiated (**HWRES** command) or can be caused by an interruption in power to the unit. Therefore, when you make changes to configuration or adjustment variables, you should save the changes to the controller's flash memory. When the controller comes back up after a hardware reset, it reads the values stored in the flash memory to RAM.

11.1 Saving Variable Values to the Controller's Flash Memory

1. In **SPiiPlus MMI → Adjuster**, click **Save To Flash.**

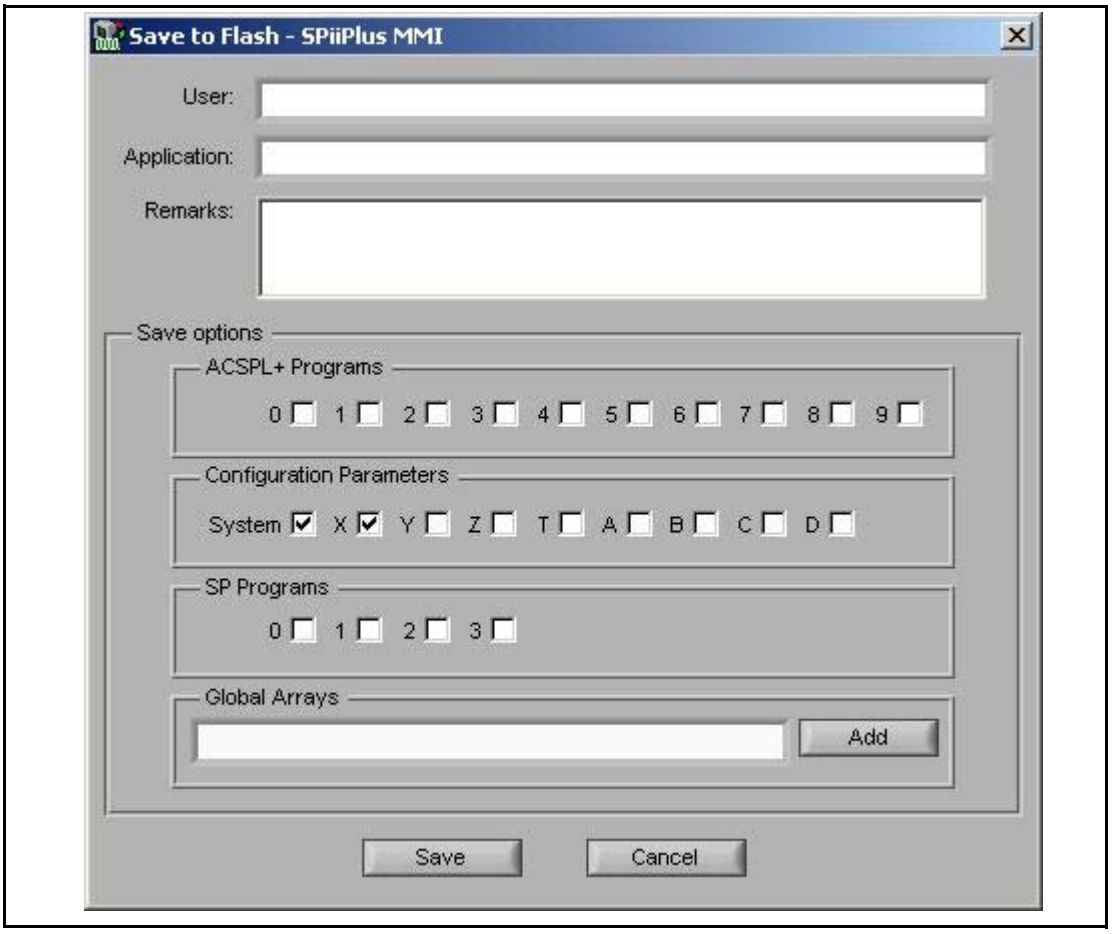


Figure 39 Save to Flash Dialog

2. In the **Configuration Parameters** group, verify that the axes for which you changed variables are selected. Selecting an axis determines that all the configuration and adjustment variable values for that axis will be saved.

3. Click **Save** to save the changed variable values.

11.2 Duplicating Configuration and Adjustment from One Axis to Another

To save time when working with identical, or similar, axes, you can copy configuration and adjustment data from an axis that you have already setup to one or more other axes. The copied data can be modified as necessary.

1. Open **MMI → Setup → Configurator** and click **Copy Axis.**

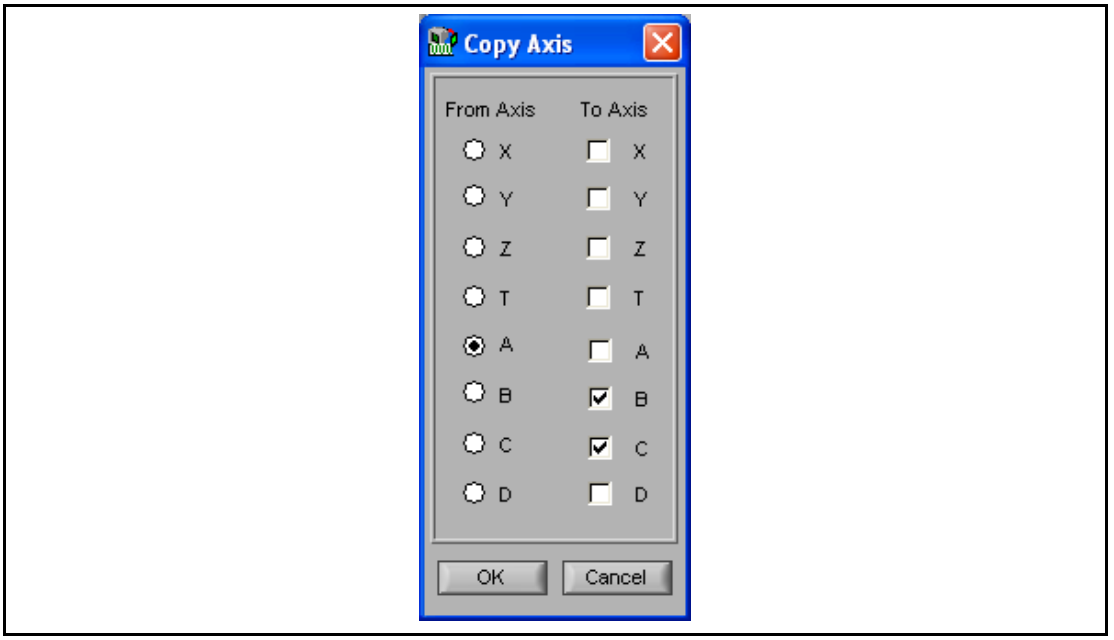


Figure 40 Copy Axis Dialog

2. Select the **From Axis** - the source axis for the configuration and adjustment data that will duplicated to other axes**.**
3. Select the **To Axis** - the target axis (or axes).
4. A confirmation message appears, informing you that:

- the source axis values will be saved to flash memory
- the target axes will be disabled
- the source axis values will be saved to the target axis flash memory and loaded to RAM

11.3 Protecting Data from Unintended Changes

Advanced	*This topic is for users who are familiar with SPiiPlus ACSPL+ programming.*

Up to this point, the controller has been in configuration mode, meaning that it can be written to. Changing the controller mode to protected limits changes that can be made to controller variables. Protected mode can be useful once you have completed configuration and adjustment and want to protect the controller from changes that could be caused unintentionally. In protected mode, the following cannot be performed:

- Changes to configuration and adjustment variables
- Modifications to user programs.
- Operations that affect the flash memory (for example, the immediate commands: **#SAVE** and **#LOAD**).

11.3.1 Activating Protected Mode

1. Open **SPiiPlus MMI → Communication Viewer**

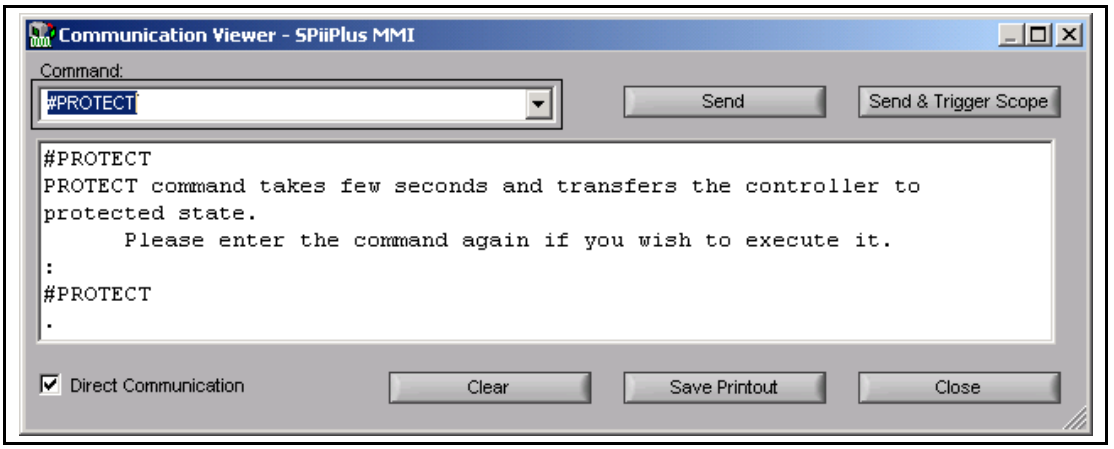


Figure 41 Communication Viewer

The **Communication Viewer** provides direct communication with the controller.

2. Enter **#PROTECT** and press **ENTER** twice.

Warning	
	***The #PROTECT command resets the SPiiPlus Controller!*** ***Any motion that is active will be terminated. Any data that has not been saved to the flash memory will be lost.***

3. Verify that Protected Mode is active by opening **SPiiPlus MMI → Application → Protection Status.**

Figure 42 Protection Status Message

11.3.2 Other Protection-Related Features

To keep a program buffer the option to change configuration parameters in protected mode, do the following before putting the controller into protected mode (in which case the **PFLAGS** parameter will be inaccessible):

1. Open **MMI → Setup → Configurator → Program Buffer Parameters**.
2. In the **Program Buffer Parameters** dialog box, select the **Program Buffer**.
3. Select the **Privileged Buffer** (bit 4).

To determine whether protection mode is active, examine the value of the **CFG** (configuration) variable:

If **CFG** = 0, controller is in protected mode.

If **CFG** = 1, controller is in configuration mode.

11.3.3 Deactivating Protected Mode

1. Open **MMI → Tools → Communication Viewer.**
2. Click **Direct Communication.**
3. Click **Clear.**
4. Enter **#UNPROTECT** and press **ENTER** twice.

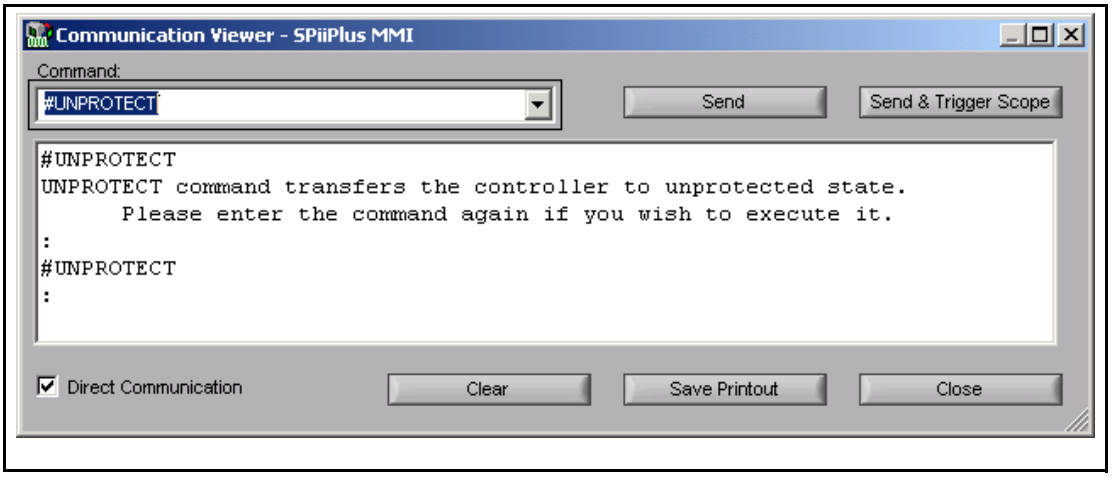


Figure 43 Communication Viewer

5. Verify that protected mode is inactive (controller is back in configuration mode) by opening **SPiiPlus MMI → Application → Protection Status**.

Figure 44 Protection Status Message

12 Homing Programs

When an incremental feedback device (for example, incremental encoder or analog input) is used, the absolute position of the axis is unknown after powerup. In such cases, a homing procedure is used.

The goal of the homing procedure is to find the absolute position of the axis based on an accurate reference point, typically an index signal or a hard stop. The absolute position of the axis is assigned to the position counter.

The homing procedure is as follows:

1. After powerup, move from current position in positive/negative direction to the end of travel (usually a limit switch or a hard stop).
2. Move backward to the first index.
3. Set the reference position (**RPOS**) to the position of the index.
4. Move to zero (or any position).

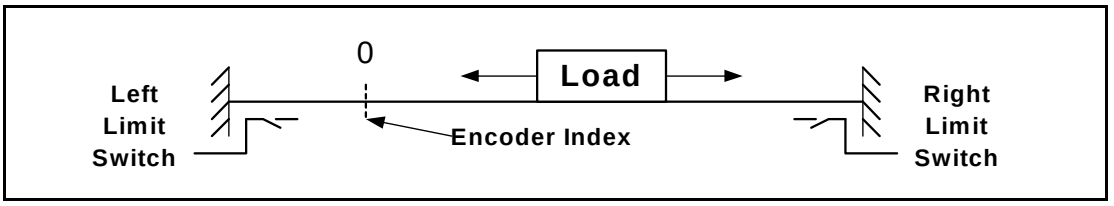


Figure 45 Homing

Note	
	1. *If the axis is equipped with a DC Brushless motor, then the homing program should also include a commutation startup program, see* ***Section 7.3 - "Commutation Startup Program"****.* 2. *The controller's programming language (ACSPL+) supports designating the axis by name or by number. This makes it possible to execute homing for all the axes using a simple loop.*

12.1 Homing Program Using Limit Switches

The following program is an example of a homing procedure for an X axis that is equipped with a DC brush motor and limit switches.

```
! The program executes the following sequence:
! - Move to the left limit switch.
! - Move to the encoder index.
! - Wait for the left limit release.
! – Set the axis origin to the position of index.
! - Move to the origin.

INT AXIS                    ! Define a variable named “AXIS”.
AXIS=0                      ! Define axis number (0=X axis, 1=Y axis...).
VEL(AXIS)= 2000             ! Set maximum velocity.
ACC(AXIS)= 100000           ! Set acceleration.
DEC(AXIS)= 100000           ! Set deceleration.
JERK(AXIS)= 20000000        ! Set jerk.
FDEF(AXIS).#LL=0            ! Disable the axis left limit default response.
FDEF(AXIS).#RL=0            ! Disable the axis right limit default response.

ENABLE (AXIS)               ! Enable the axis drive.
JOG (AXIS),-                ! Move to the left limit switch.
TILL FAULT(AXIS).#LL        ! Wait for the left limit switch activation.
JOG (AXIS),+                ! Move to the encoder index.
TILL ^FAULT(AXIS).#LL       ! Wait for the left limit release.
IST(AXIS).#IND=0            ! Reset the index flag.
TILL IST(AXIS).#IND         ! Wait till motor reaches the index.
SET FPOS(AXIS)=FPOS(AXIS)-IND(AXIS)
                            ! Set axis origin to the position of the index.
PTP (AXIS),0                ! Move to the origin.
FDEF(AXIS).#LL=1            ! Enable the axis left limit default response
FDEF(AXIS).#RL=1            ! Enable the axis right limit default response
STOP
```

12.2 Homing Program Using Hard Stops

The following program is an example for a homing procedure of X axis that is equipped with a DC brush motor and a hard stop (no limit switches).

```
! The program executes the following sequence:
! - Move to the left hard stop.
! - Move to the encoder index.
! - Set the axis origin to the position of index.
! - Move to the origin.

GLOBAL INT AXIS            ! Define a global variable named “AXIS”.
AXIS=0                     ! Define axis number (0=X axis, 1=Y axis...).
VEL(AXIS)= 500             ! Set maximum velocity.
ACC(AXIS)= 50000           ! Set acceleration.
DEC(AXIS)= 50000           ! Set deceleration.
JERK(AXIS)= 10000000       ! Set jerk.
KDEC(AXIS)= 200000         ! Set kill deceleration.
FDEF(AXIS).#CPE=0          ! Disable the default response of critical
                           ! position error fault.
XCURV(AXIS)=???            ! XCURV is used to limit the force applied to hard
                           ! stop (Define ??? per case.)
ENABLE (AXIS)! Enable the axis drive.
JOG (AXIS),-               ! Move slowly to the left hard stop.
TILL ABS(X_PE)>???         ! Identify the hard stop by the position
                           ! error.(Define??? per case.)
JOG (AXIS),+               ! Move to the encoder index.
IST(AXIS).#IND=0           ! Reset the index flag - activate index circuit.
TILL IST(AXIS).#IND        ! Wait for crossing the index.
SET FPOS(AXIS)=FPOS(AXIS)-IND(AXIS)
                           ! Set axis origin to the position of the index.
PTP (AXIS),0               ! Move to the origin.
FDEF(AXIS).#CPE=0          ! Enable the default response of critical position
                           ! error fault.

STOP
```

13 Appendix A: Nanomotion Piezo Ceramic Motors

This appendix discusses Nanomotion piezo ceramic motors driven by ab1,ab2 and ab4 amplifiers. When adjusting a motor driven by the ab5 amplifier, adjust the motor as a regular DC motor (do not enable **MFLAGS** bit 7).

Enable NanoMotion support in one of two ways:

- **SPiiPlus MMI → Setup → Adjuster → Position Loop Adjustment → NanoMotion → Enable NanoMotion (Figure 46)**
- **SPiiPlus MMI → Setup → Configurator → Motor Flags (MFLAGS) → NanoMotion Piezoceramic Motor (7)**.

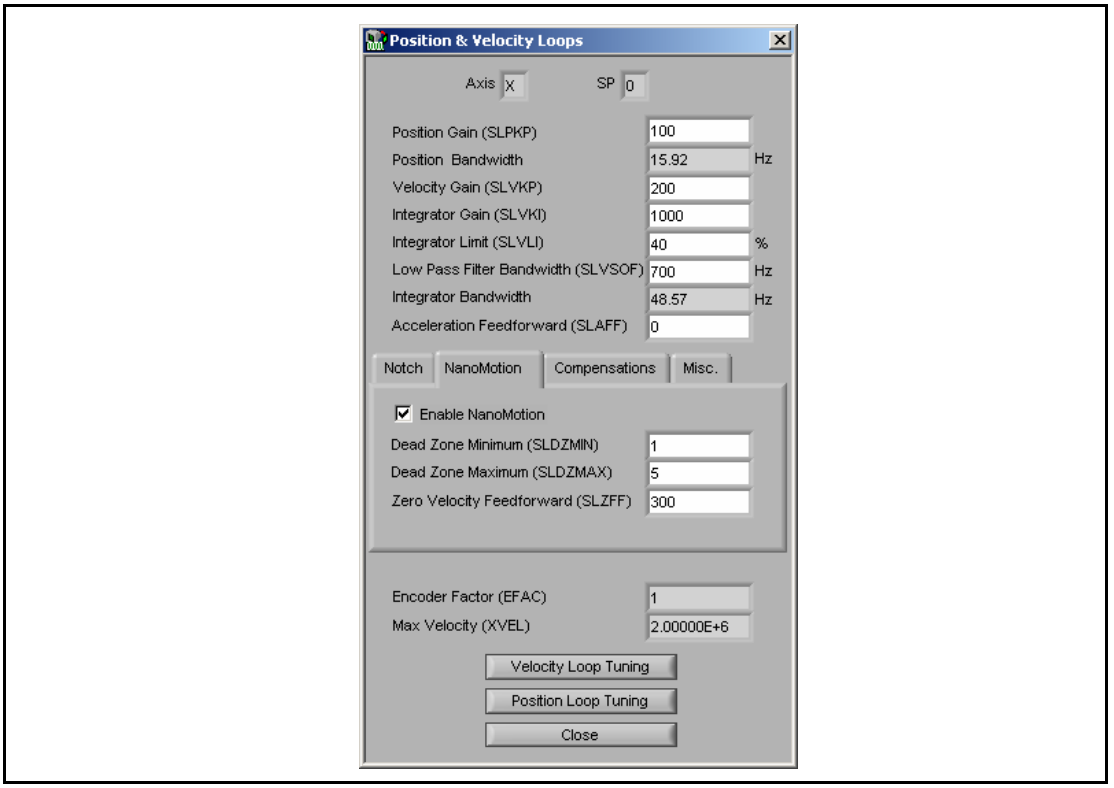


Figure 46 NanoMotion Tab in Position and Velocity Loops Dialog Box

13.1 Dead Zone Mechanism

This mechanism improves the settling time by taking advantage of the intrinsic motor friction. The algorithm stops the motor when the position approaches the target within the Dead-Zone Minimum (**SLDZMIN** parameter) in user units. When the error radius increases above Dead-Zone Maximum (**SLDZMAX** parameter) the motor is "servoing" again.

The parameter values depend on the system specifications. Usually:

- **SLDZMIN<*axis*>** is between 1.0 to 2.0 counts
- **SLDZMAX<*axis*>** is between 4.0 to 10.0

13.1.1 Dead Zone Example

Figure 47, shows the feedback position - **FPOS** (green), the drive output - **DOUT** (red) and the position error - **PE** (white).

It can be seen that when the position error is lower than the **SLDZMIN** parameter the controller cuts the servo by turning off the drive-output. The motor then stays in place due to the intrinsic friction (with zero jitter).

When the position error goes above the **SLDZMAX** parameter the drive output is turned on again.

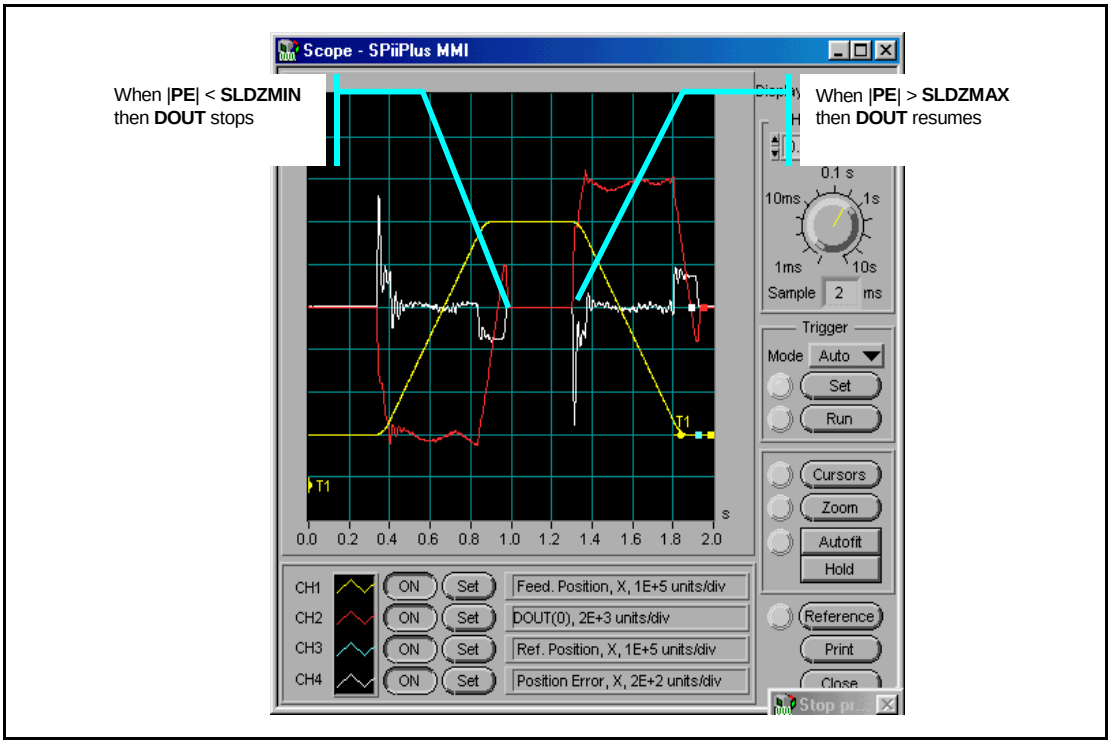


Figure 47 Dead Zone Mechanism

13.2 Zero Velocity Feed Forward

The zero velocity feed forward mechanism improves settling time by stopping the feed forward velocity when the axis is getting close to the target position. The distance from the target is defined by the variable **SLZFF** (in user units). The proper value of **SLZFF** depends on the total moving mass and the resolution of the encoder. It increases with mass and encoder resolution. Usually:

For HR1 motor with encoder resolution of 0.1µM, set **SLZFF** to 100 – 300 counts.

For HR8 motor with encoder resolution of 0.1µM, set **SLZFF** to 300 - 400 counts.

13.2.1 Zero Velocity Feed Forward Example

A comparison of **Figure 47** (**SLZFF**=0) with **Figure 48** (**SLZFF**=300 counts) shows a significant improvement in settling time from 900 [msec] to 6 [msec].

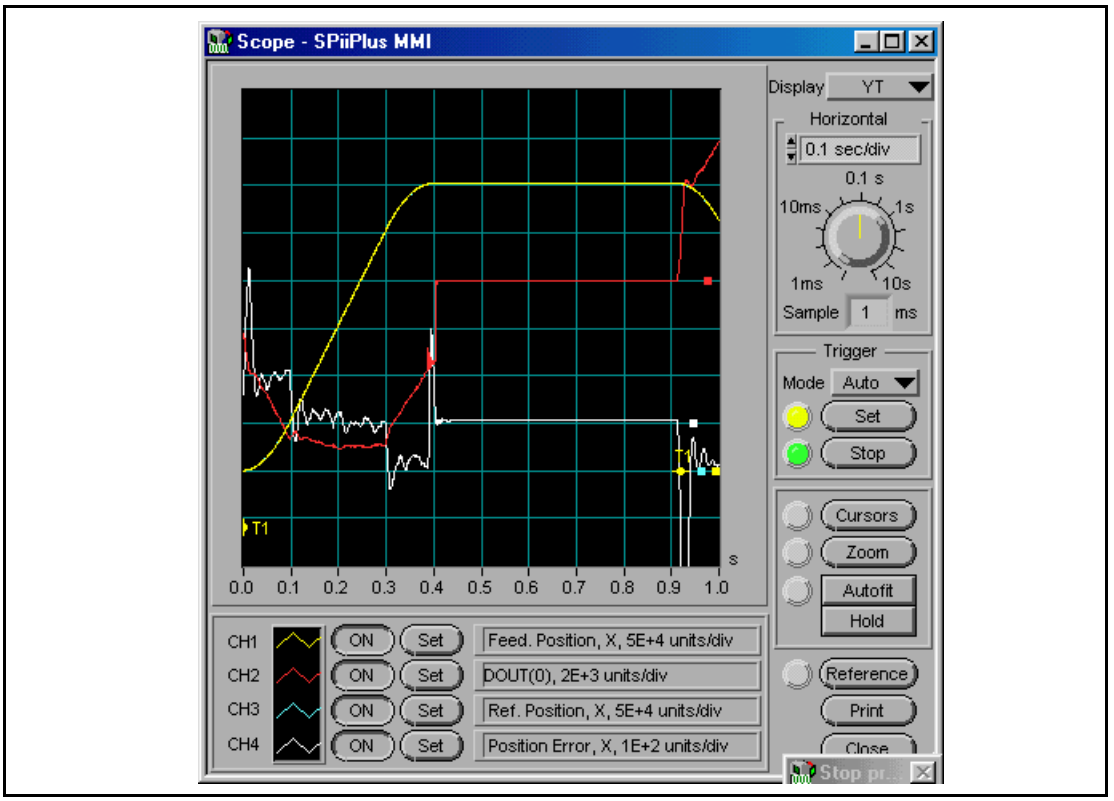


Figure 48 Improved Settling Time Using Zero Velocity Feed Forward

The following figures shows the reference position (blue) and feedback position (green) with **SLZFF**=0 and with **SLZFF**=300 counts (**Figure 49** and **Figure 50**). The significant improvement in settling time can be clearly observed. With **SLZFF**=0 the response has some overshoot and the feedback position settles to the desired one from above. With the **SLZFF**=300 counts, there is no overshoot and the feedback position settles to the desired one

from below.

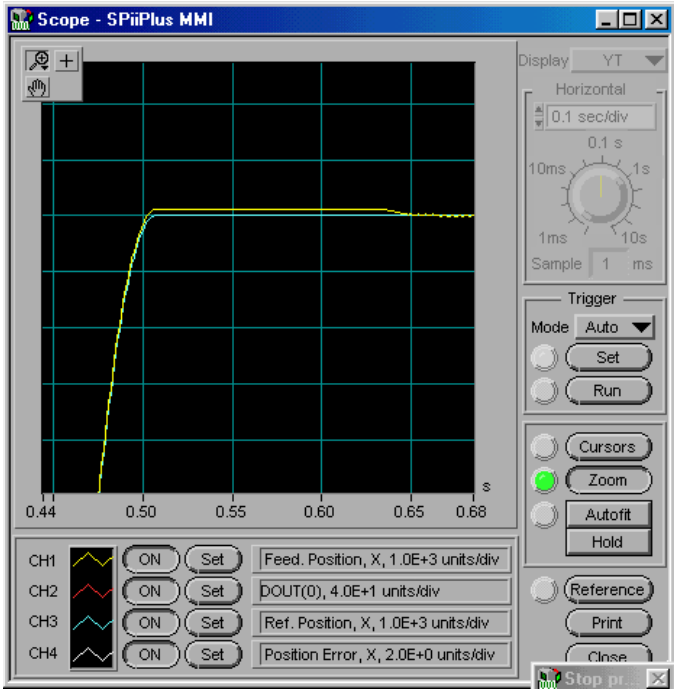


Figure 49 Settling with SLZFF=0

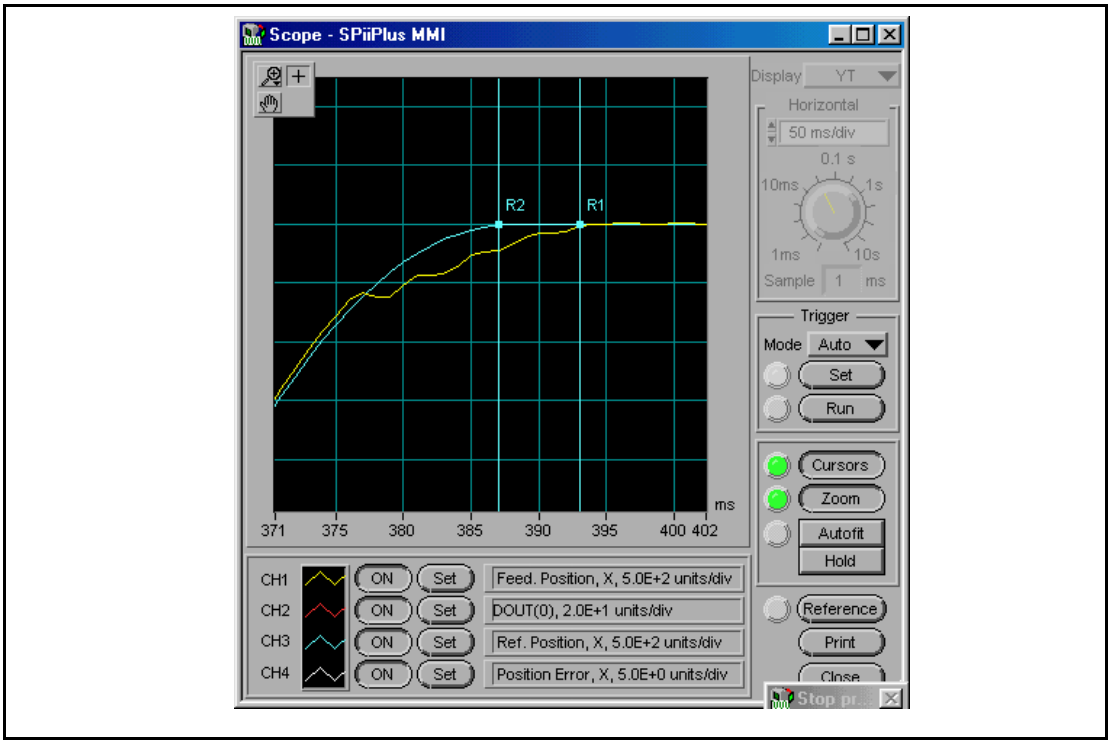


Figure 50 Settling with SLZFF=300 counts

13.3 Tuning Procedure

1. In the **Safety Parameters** dialog box, set an appropriate value to the maximum velocity parameter, **XVEL**. This parameter is also used for determining the velocity scale factor: the higher its value, the lower the scale factor. If **XVEL** is not large enough, the value of the **SLVKP** parameter will be relatively small.
2. In Axis Setup define the motor as DC brush and the amplifier as DC Brush, 1 input.
3. In Open Loop Verification verify the correct polarity of the drive output.
4. In Position Loop Adjustment, enable the NanoMotion support check box.
5. Set SLDZMIN to 2 counts and SLDMAX to 10 counts.
6. Set SLZFF parameter to 300 counts.
7. Adjust Velocity and position loops. Typical value of integrator (SLVKI) is ~1000.
8. Optimize SLZFF, SLDZMIN and SLDZMAX parameters.
9. Gradually increase the static friction compensation SLFRC to reduce position error during motion startup.
10. Note: Set the target radius parameter TARGRAD (in the Configurator) equal to SLDZMAX.

14 Appendix B: Netzer Precision Electric EncoderS™

14.1 Overview

Netzer Precision Motion Sensors Ltd. (http://www.netzerprecision.com/) develops and produces advanced absolute sin-cos encoders. This technical application note describes how to use the SPiiPlus motion controller with sin-cos feedback from a rotary or linear Netzer Electric Encoder[tm].

Note	
	*The following description relates to rotary encoders. In the case of linear encoders, substitute distance for revolution. For example, instead of the ECR (electrical cycle per revolution) use electrical cycles per distance (mm or inch).*

Netzer encoders have two modes of operation:

- **Coarse mode** – Provides one ECR (electrical cycle per revolution), which enables absolute position reading over a full revolution.
- **Fine mode –** Generates multiple ECRs. Although the encoder produces a relatively small number of ECRs, using the SPiiPlus encoder's optional internal sin-cos multiplier, high feedback resolution can be achieved.

The mode of operation is selected using one of the SPiiPlus controller's digital outputs. On powerup, the encoder's coarse mode is used to identify the absolute position. Then the encoder is switched to fine mode for high-resolution operation.

If a software-commutated DC brushless motor is used, then commutation has to be done only once: at initial setup. On subsequent powerups the commutation phase will be retrieved based on the absolute position.

14.2 Netzer Precision ACSPL+ Programs

There are two ACSPL+ programs to be used with a Netzer encoder:

- **Netzer_Startup.prg**
  - Motor Type: Only where software commutation is required (AC servo/DC brushless motors).
  - Execution: Once only: after Commutation Adjustment setup.

  The program calculates the offset between the absolute position and the commutation phase and stores it in the **SLCORG** (commutation origin) variable. After executing the program, you must save **SLCORG** to the controller's flash memory.

- **Position_Retrieval.prg**
  - Motor Type: For every type of motor, including (AC servo/DC brushless motors).
  - **Execution**: After every powerup.

  During the program's execution, the motor is disabled. The program retrieves the absolute position of the axis. If the controller's software commutation capability is used (AC servo/DC brushless motors), the program also retrieves the commutation phase (according to the absolute position and the **SLCORG** variable).

14.2.1 Hardware Setup

1. The encoder's differential sin-cos signals should be directly connected to two analog inputs of the SPiiPlus controller.
2. The encoder's course/fine select (C/F select) wire should be connected to one of the digital outputs of the SPiiPlus controller. This way the encoder can be switched between the two modes by simply changing the value of the bit of the **OUT** variable for the connected output.

   For example, if the C/F select wire is connected to digital output #0:

   - `OUT0.0 = 0` sets the encoder to coarse mode
   - `OUT0.0 = 1` sets the encoder to fine mode

   Switching between the two modes is done with OUT0.0 in the appended programs.

14.2.2 Software Setup for First Usage

1. Load `Position_Retrieval.prg` to a buffer in the controller. If the motor is software-commutated AC servo/DC brushless, then load **Netzer_Startup.prg** to a separate buffer.
2. Open **SPiiPlus MMI → I/0 Monitor**. The I/O are represented by interactive indicators.
3. If the indicator for the digital output connected to the C/F select wire is not on (), click on it to turn it on (). The sets the related variable, which in turn activates the output, setting the encoder to fine mode.
4. Open **SPiiPlus MMI → Adjuster → Safety Parameters**. Verify that all safety variables are defined for the desired resolution.
5. Open **SPiiPlus MMI → Adjuster → Axis Setup**. Set the **Drive** and **Motor** variables.
6. On the **Position Feedback** tab, select **Sin-Cos** in the **Type** field.
7. Enter the number of Fine Mode ECRs in the **Lines per Revolution** field**.**
8. Select the desired **Internal Multiplier Factor** (**E_SCMUL** variable) for the required position resolution.

Note	
	*The resulting* **Counts per Revolution** = ECRs x 2^{E_SCMUL}. *For example, with a rotary motor where the encoder has 8 ECRs in fine mode and the multiplier is 12, we get 32,768 counts per revolution (8 x 2^{12}). Maximum resolution can be achieved with multiplier = 16, which yields 524,288 counts per revolution.*

9. If software commutation is required (AC servo/DC brushless motor), open **SPiiPlus MMI ➔ Adjuster ➔ Commutation ➔ Preferences.** (The following are required only once, at initial setup).
   a) Verify that **A Detent Poin**t is selected for the **Retrieve Commutation Phase at** field.
   b) In the Commutation dialog box, select **Start Commutation Program** to perform commutation.
   c) Execute the ACSPL+ program: `Netzer_Startup.prg`.
10. The `Position_Retrieval.prg` program is set to work with a rotary motor. In the case of a linear motor, change the **ENC_RES** variable to equal the total encoder length in fine mode (counts). Run `Position_Retrieval.prg`**.**
11. The program has an offset variable **ABS_OFFS**, which enables you to define any point as a zero position. For example, when the feedback position = 1000, setting **ABS_OFFS** = 1000 and running the program again, while the motor is standing still, will define the current position as zero.
12. Save all variables and `Position_Retrieval.prg` to the controller flash memory.
13. Complete all other adjustment steps and again save all variables to the flash memory.

14.2.3 Subsequent Powerups

Execute the ACSPL+ program, **Position_Retrieval.prg** after any power-up, with the motor disabled. It is recommended to insert the program into an AUTOEXEC routine, so that it will be automatically activated on startup.

14.2.4 Netzer_Startup.prg

```
!For Using Netzer Precision's Sin-Cos Electric Encoder with a SPiiPlus
!Controller
!*************************************************************************
!  FW version: ???
!*************************************************************************
!  Remark: Coarse/ fine selection wire is connected to digital output #0
!*************************************************************************

REAL AXIS                   ! Axis number
REAL ABS_POS                ! Absolute position (2^16 resolution)
REAL ABS_PHASE              ! Absolute phase (elec. deg)
REAL COMM_PHASE             ! Reconstructed commutation phase
REAL COMM_ORG               ! Commutation origin
AXIS = 0
OUT0.0 = 0                  ! Operate in coarse mode
WAIT 500
DISABLE(AXIS)
! Get absolute position from SPii (2^16 resolution)

if (AXIS)=0; ABS_POS  = GETSPV(SP0:X_POS_ATAN) ; end
if (AXIS)=1; ABS_POS  = GETSPV(SP1:X_POS_ATAN) ; end
if (AXIS)=2; ABS_POS  = GETSPV(SP2:X_POS_ATAN) ; end
if (AXIS)=3; ABS_POS  = GETSPV(SP3:X_POS_ATAN) ; end
if (AXIS)=4; ABS_POS  = GETSPV(SP0:A_POS_ATAN) ; end
if (AXIS)=5; ABS_POS  = GETSPV(SP1:A_POS_ATAN) ; end
if (AXIS)=6; ABS_POS  = GETSPV(SP2:A_POS_ATAN) ; end
if (AXIS)=7; ABS_POS  = GETSPV(SP3:A_POS_ATAN) ; end

OUT0.0=1                        ! Operate in fine mode
WAIT 500
! Calculate Commutation Origin
COMM_PHASE = getconf(214,AXIS)    ! Get commutation phase
ABS_PHASE = (ABS_POS/POW(2,16)-FLOOR(ABS_POS/POW(2,16)))*180*SLCNP(AXIS)
If MFLAGS(AXIS).13              ! If drive output is inverted
COMM_ORG  = COMM_PHASE + ABS_PHASE

else                            ! Drive output is not inverted
COMM_ORG = COMM_PHASE - ABS_PHASE
end
SLCORG(AXIS)= COMM_ORG - FLOOR(COMM_ORG/360)*360
Stop
```

14.2.5 Position_Retrieval.prg

```
!For Using Netzer Precision's Sin-Cos Electric Encoder with a SPiiPlus
!Controller
!*************************************************************************
!  FW version: ???
!*************************************************************************
!  Remark: Coarse/ fine selection wire is connected to digital output #0
!*************************************************************************

REAL ABS_POS               ! Absolute position (2^16 resolution)
EAL AXIS                   ! Axis number
REAL ABS_FPOS              ! Absolute position (user units)
REAL ABS_PHASE             ! Absolute phase (elec. deg)
REAL COMM_PHASE            ! Retrieve commutation phase
REAL ABS_OFFS              ! Absolute position offset (user defined)
REAL ENC_RES               ! Encoder resolution (counts per rev/length)

!*************************************************************************
! POSITION RETRIEVAL
!*************************************************************************

AXIS = 0
OUT0.0=0                   ! Operate in coarse mode
WAIT 500
DISABLE(AXIS)

! Get absolute position from SPii (2^16 resolution)
if (AXIS)=0; ABS_POS  = GETSPV(SP0:X_POS_ATAN) ; end
if (AXIS)=1; ABS_POS  = GETSPV(SP1:X_POS_ATAN) ; end
if (AXIS)=2; ABS_POS  = GETSPV(SP2:X_POS_ATAN) ; end
if (AXIS)=3; ABS_POS  = GETSPV(SP3:X_POS_ATAN) ; end
if (AXIS)=4; ABS_POS  = GETSPV(SP0:A_POS_ATAN) ; end
if (AXIS)=5; ABS_POS  = GETSPV(SP1:A_POS_ATAN) ; end
if (AXIS)=6; ABS_POS  = GETSPV(SP2:A_POS_ATAN) ; end
if (AXIS)=7; ABS_POS  = GETSPV(SP3:A_POS_ATAN) ; end
OUT0.0=1                                ! Operate in fine mode

WAIT 500
! Convert absolute position to user units:
! In case of rotary motor ENC_RES = SLCPRD(AXIS)
! In case of linear motor ENC_RES = counts per total encoder length (in fine
mode)
ENC_RES = SLCPRD(AXIS)     ! Change in case of linear motor
ABS_FPOS = (ABS_POS/POW(2,16)-FLOOR(ABS_POS/POW(2,16)))*ENC_RES*EFAC(AXIS)
                           ! ABS_OFFS is an offset in absolute position
! Set it to define required "zero position"
ABS_OFFS = 0
SET FPOS(AXIS) = ABS_FPOS - ABS_OFFS
!*************************************************************************
```

```
! COMMUTATION RETRIEVAL
! This section is relevant only to software commutated brushless motor and
can be omitted in case of DC brush motor or hardware commutation.
!*************************************************************************

! Convert Absolute Position to Electrical Degrees:
ABS_PHASE = (ABS_POS/POW(2,16)-FLOOR(ABS_POS/POW(2,16)))*180*SLCNP(AXIS)
! Reconstruct Commutation Phase
If MFLAGS(AXIS).13                      ! If drive output is inverted
COMM_PHASE = SLCORG(AXIS)- ABS_PHASE
else                                    ! drive output is not inverted
COMM_PHASE = SLCORG(AXIS)+ ABS_PHASE
end
SETCONF(214,(AXIS),COMM_PHASE)
MFLAGS(AXIS).9=1                        ! Set commutation flag
stop
```

15 Appendix C: Working with a Gantry Axis

15.1 Overview

Gantry motion systems incorporate two or more motors, which work simultaneously to move an axis. Gantry axes provide flexible and efficient solutions for a wide range of applications like: inspection systems, materials handling, pick and place, machine loading and unloading etc.

Figure 51 illustrates a typical gantry system with two motors (X1, X2) that move a single axis (X):

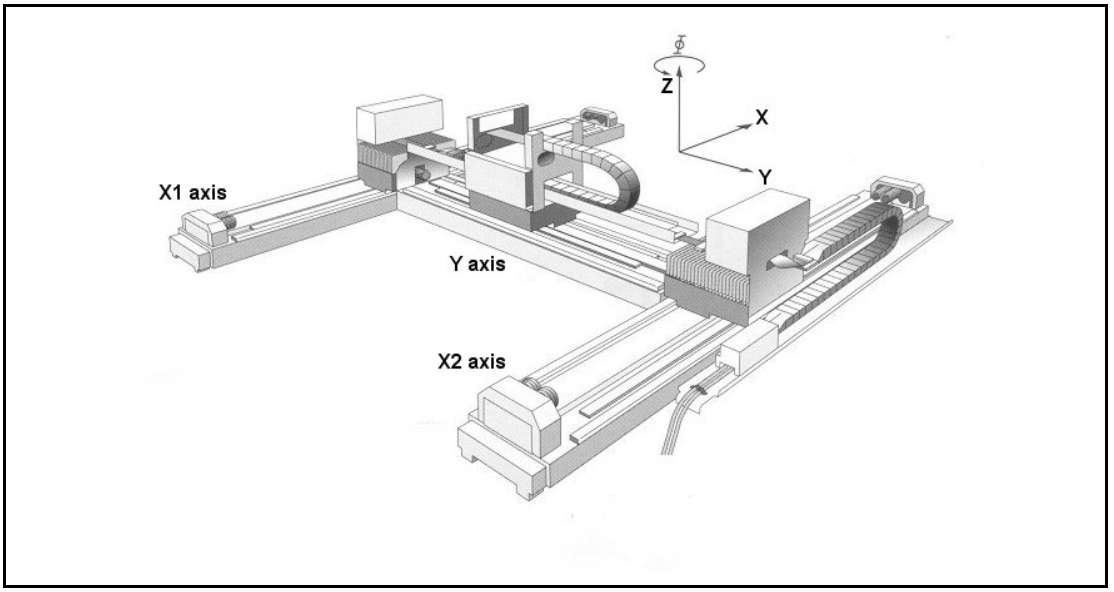


Figure 51 Typical Gantry

15.2 Gantry Implementation Using Connect And Depends Commands

Implementing complex motions like gantry is made easy with the SPiiPlus axis profile generation commands: **connect** and **depends** commands. In a gantry application, the **connect** command is used to define the relation between two variables: **RPOS** (reference axis position) and **APOS** (dummy axis position). By default, the relation between **RPOS** and **APOS** is 1:1.

In a gantry application the **connect** command enables generation of an identical motion profile for two (or more) motors – referred to as master and slave motors (see next figure). The **connect** command creates and maintains a direct link (like a mechanical link) between the motion profile of the master motor to the slave motor.

Once a **connect** command has defined a new relation for **RPOS**, the relation remains active until:

- A different **connect** relation is defined
- The controller is shutdown
- HWRESET/RESET commands are executed

The following block diagram demonstrates the axis profile generation for gantry with two motors:

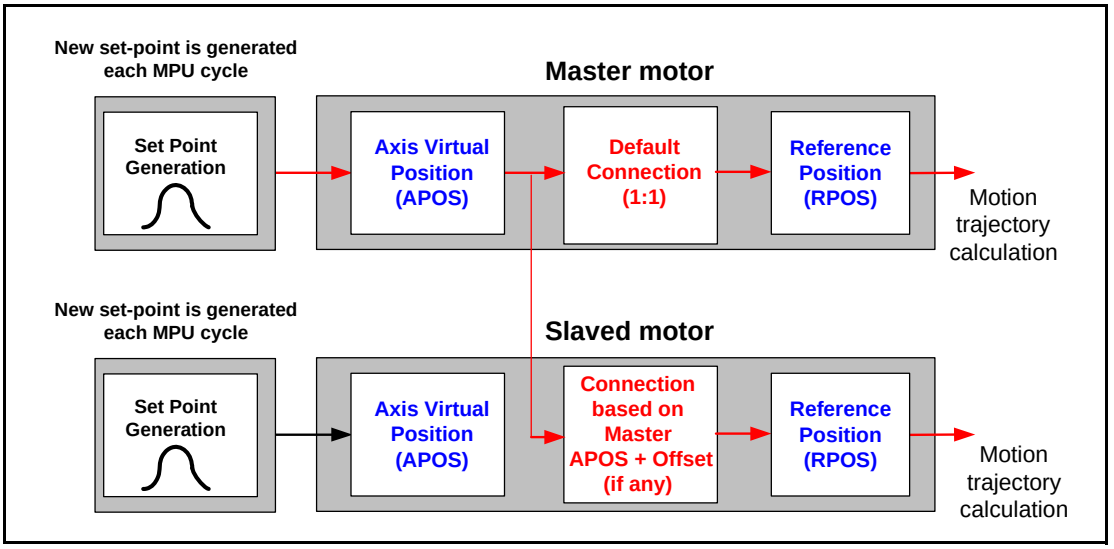


Figure 52 Gantry Axis Profile Generation

The **DEPENDS** command specifies a logical dependence between a motor and axes. By default the motor is assigned to its axis. **DEPENDS** is necessary whenever the **CONNECT** command is used.

The **SAFETYGROUP** command that provides a special operation and safety mechanism for gantry implementation as follows:

SAFETYGROUP ***(master axis, slaved axis, slaved axis…)***
This command executes the following actions:

Copies the fault configuration from the first specified axis (master) to all subsequent axes.

Blocks the axes together in response to **KILL** and **DISABLE** operations.

After the command, all specified axes are killed or disabled at once, even if the operation is directed to one of the axes. This applies to both the operation initiated by the commands or by the faults.

15.3 Gantry Implementation

The following implementation is for a gantry application involving two identical motors and feedback devices. Similar implementations can be derived for gantry applications involving more axes (one master motor and two or more slave motors).

Preparation for gantry implementation comprises the following steps:

1. Gantry axes setup, configuration and adjustment
2. Gantry axes commutation and homing

Once these steps have been accomplished the system is ready for normal gantry operation. Meaning, the user needs to refer to the master axis only and the slaved axis will be operated automatically.

15.3.1 Gantry Setup and Configuration

1. **MMI → Setup → Configurator** : Set the safety parameters for the master (limits, current values, position error etc). Copy the configuration to the identical slaved axis – see **Section 11.2 - "Duplicating Configuration and Adjustment from One Axis to Another"**.
2. **MMI → Setup → Adjuster → Commutation**: make commutation separately for each (if DC brushless motor is being used).
3. **MMI → Setup → Adjuster → Open loop verification**: make verification in open loop separately for each motor.
4. Load the following program to a buffer and run it:

```
! THIS PROGRAM IS USED TO ADJUST A GANTRY SYSTEM
INT GLOBAL M_AXIS, S_AXIS ,BUFFER !"M" FOR MASTER AXIS, "S" FOR SLAVED AXIS
M_AXIS=4 ; S_AXIS=5 ;BUFFER=7! SET HERE THE INVOLVED AXIS NUMBERS A=4, B=5
DISABLE(M_AXIS);DISABLE(S_AXIS)
!------ Configuration variables ---------------------
ERRI(S_AXIS)=ERRI(M_AXIS)
ERRV(S_AXIS)=ERRV(M_AXIS)
ERRA(S_AXIS)=ERRA(M_AXIS)
CERRI(S_AXIS)=CERRI(M_AXIS)
CERRV(S_AXIS)=CERRV(M_AXIS)
CERRA(S_AXIS)=CERRA(M_AXIS)
DELI(S_AXIS)=DELI(M_AXIS)
DELV(S_AXIS)=DELV(M_AXIS)
SLLIMIT(S_AXIS)=SLLIMIT(M_AXIS)
SRLIMIT(S_AXIS)=SRLIMIT(M_AXIS)
XVEL(S_AXIS)=XVEL(M_AXIS)
XACC(S_AXIS)=XACC(M_AXIS)
VELBRK(S_AXIS)=VELBRK(M_AXIS)
XRMS(S_AXIS)=XRMS(M_AXIS)
XRMST(S_AXIS)=XRMST(M_AXIS)
XCURI(S_AXIS)=XCURI(M_AXIS)
XCURV(S_AXIS)=XCURV(M_AXIS)
```

```
!------ Adjustment variables ----------------------
SLPKP(S_AXIS)=SLPKP(M_AXIS)
SLVKP(S_AXIS)=SLVKP(M_AXIS)
SLVKI(S_AXIS)=SLVKI(M_AXIS)
SLVLI(S_AXIS)=SLVLI(M_AXIS)
SLVSOF(S_AXIS)=SLVSOF(M_AXIS)
SLIOFFS(S_AXIS)=SLIOFFS(M_AXIS)
SLFRC(S_AXIS)=SLFRC(M_AXIS)
SLAFF(S_AXIS)=SLAFF(M_AXIS)
SLCPA(S_AXIS)=SLCPA(M_AXIS)
EFAC(S_AXIS)=EFAC(M_AXIS)

!------ Motion variables ---------------------
VEL(S_AXIS)=VEL(M_AXIS)
ACC(S_AXIS)=ACC(M_AXIS)
DEC(S_AXIS)=DEC(M_AXIS)
JERK(S_AXIS)=JERK(M_AXIS)
STOP

! These auto routines update the adjustment variables
! of the S_AXIS to match the M_AXIS
ON SLPKP(S_AXIS)<>SLPKP(M_AXIS); SLPKP(S_AXIS)=SLPKP(M_AXIS); RET
ON SLPKP(S_AXIS)<>SLPKP(M_AXIS); SLPKP(S_AXIS)=SLPKP(M_AXIS); RET
ON SLVKP(S_AXIS)<>SLVKP(M_AXIS); SLVKP(S_AXIS)=SLVKP(M_AXIS); RET
ON SLVKI(S_AXIS)<>SLVKI(M_AXIS); SLVKI(S_AXIS)=SLVKI(M_AXIS); RET
ON SLVLI(S_AXIS)<>SLVLI(M_AXIS); SLVLI(S_AXIS)=SLVLI(M_AXIS); RET
ON SLVSOF(S_AXIS)<>SLVSOF(M_AXIS); SLVSOF(S_AXIS)=SLVSOF(M_AXIS); RET
ON SLIOFFS(S_AXIS)<>SLIOFFS(M_AXIS); SLIOFFS(S_AXIS)=SLIOFFS(M_AXIS); RET
ON SLFRC(S_AXIS)<>SLFRC(M_AXIS); SLFRC(S_AXIS)=SLFRC(M_AXIS); RET
ON SLAFF(S_AXIS)<>SLAFF(M_AXIS); SLAFF(S_AXIS)=SLAFF(M_AXIS); RET
ON SLCPA(S_AXIS)<>SLCPA(M_AXIS); SLCPA(S_AXIS)=SLCPA(M_AXIS); RET
ON EFAC(S_AXIS)<>EFAC(M_AXIS); EFAC(S_AXIS)=EFAC(M_AXIS); RET
ON XVEL(S_AXIS)<>XVEL(M_AXIS); XVEL(S_AXIS)=XVEL(M_AXIS); RET
```

5. Open **MMI ➔ Adjuster ➔ Position and Velocity Loop**: adjust the maser axis normally. The slaved axis parameters will be set identically (by the autoroutines in the program) to the master axis.

15.3.2 Commutation and Homing

Load the following program to a buffer and run it:

```
!Commutation and homing for a gantry axes
! ------------ Define axes variables -------------------------
INT GLOBAL M_AXIS; M_AXIS=4   ! Define Master axis (A axis in this example)
INT GLOBAL S_AXIS; S_AXIS=5   ! Define Slaved axis (B axis in this example)
VEL(M_AXIS)= 4000             ! Set maximum velocity
ACC(M_AXIS)= 40000            ! Set acceleration
DEC(M_AXIS)= 40000            ! Set deceleration
JERK(M_AXIS)= 400000          ! Set jerk
SAFETYGROUP(M_AXIS,S_AXIS)

!-------- Doing commutation for a brushless Master motor--------
! Use this section only for a brushless Master motor
!ENABLE (M_AXIS)
!COMMUT (M_AXIS)
!DISABLE (M_AXIS) !To allow slight movement of the S_AXIS during COMMUT
!-------- Doing commutation for a brushless Slaved motor--------
! Use this section only for a brushless Slaved motor
!ENABLE (S_AXIS)
!COMMUT (S_AXIS)

!------------Define connect functions -------------------------
MFLAGS(M_AXIS).17=0 ;MFLAGS(S_AXIS).17=0
                                ! Turns off the Connect function default
                                ! (FPOS = APOS)
CONNECT RPOS(M_AXIS)=APOS(M_AXIS)
DEPENDS (M_AXIS),(M_AXIS)       !Master motor depends on Master axis
CONNECT RPOS(S_AXIS) = APOS(M_AXIS)
DEPENDS (S_AXIS),(M_AXIS)        !Slaved motor depends on Master axis
SET APOS(M_AXIS)=0,RPOS(M_AXIS)=0, RPOS(S_AXIS)=0


!------------Homing procedure ----------------------------------
FDEF(M_AXIS).#LL=0 ; FDEF(S_AXIS).#LL=0 ! Disable the default response of the
limit fault
ENABLE (M_AXIS); ENABLE(S_AXIS)
JOG (M_AXIS),-                ! Move to the left limit switch
TILL FAULT(M_AXIS).#LL        ! Wait for the left limit switch activation
! Can be written also as "TILL ABS(X_PE)>???" when no limit switches exist -
only a hard stop.
JOG (M_AXIS),+                ! Move to the encoder index
TILL ^FAULT(M_AXIS).#LL       ! Wait for the left limit release
IST(M_AXIS).#IND=0           ! Reset the index flag - activate index circuit
TILL IST(M_AXIS).#IND         ! Wait for crossing the index
KILL(M_AXIS,S_AXIS)
TILL ^AST(M_AXIS).#MOVE
SET FPOS(M_AXIS)=FPOS(M_AXIS)-IND(M_AXIS),
APOS(M_AXIS)=RPOS(M_AXIS),RPOS(S_AXIS)=RPOS(M_AXIS)  ! Set axis origin to the
position of index = zero
FDEF(M_AXIS).#LL=1 ; FDEF(S_AXIS).#LL=1
PTP(M_AXIS),0                 ! Move to the origin
STOP
```

15.3.3 Gantry Operation

1. After commutation and homing, issue motion commands for the master axis only, for example:
   `PTP (M_AXIS), 10000`. The slave axis will move automatically.
2. **Enable** commands should always be declared for both axes, for example:
   `ENABLE (M_AXIS); ENABLE(S_AXIS)`

16 Appendix D: Working with Dynamic Braking

16.1 About Dynamic Braking

Model	*Dynamic braking is supported only by SPiiPlus CM only (for X, Y or A axis).*

Dynamic braking is used to decelerate or eliminate motion of the axis while the drive is disabled. For example the dynamic brake can decelerate the motion in a case where a critical fault has disabled the drive during motion. Dynamic braking is executed by short-circuiting the motor phases in the drive. In the case of a three-phase motor, this is done by simultaneously activating the three lower transistors in the drive, as shown in **Figure 53**:

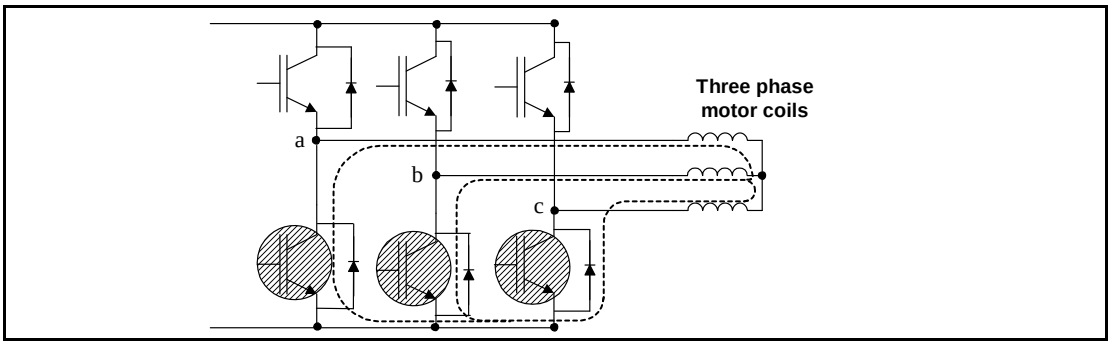


Figure 53 Dynamic Brake Implementation By The Digital Drive

In the case of a DC motor, two lower transistors are activated, short-circuiting the motor armature.

During braking, the kinetic energy stored in the total moving mass / inertia of the motor is dissipated as heat on the short-circuited phases and transistors.

Caution	***Before implementing dynamic braking, follow the procedures described in this chapter to prevent any damage to the control module, the motor, or the rest of the system. In addition, we recommend measuring the actual developed current during maximum deceleration using:*** **MMI → Tools → SPii Scope**.

16.2 Dynamic Brake - Safety Verification

16.2.1 Dissipation And Peak Current – Drive Verification

You must verify that the maximum current during the braking process is lower than the peak current of the drive.

Note	*All of the following formulas are designed for Star Connection configurations, only.*

The equation for maximum current calculation during braking process is for a linear motor is:

(1)

$$I_{DB\_\max} = \frac{K_E \cdot V_{\max}}{\sqrt{3} \cdot |Z|}$$

where:

K_E	[V·s/m] - is the phase-to-phase back EMF constant
V_{max}	[m/s] - is the maximal linear velocity of the motor
$Z=R+j\ x\ X,\ [\Omega]$	is the phase impedance (the sum of a phase resistance and phase reactance).

For a rotary motor: (2)

$$I_{DB\_\max} = \frac{K_E \cdot \omega_{\max}}{\sqrt{3} \cdot |Z|}$$

where:

K_E	[V·s/rad] - is the phase-to-phase back EMF constant
ω_{max}	[rad/s] - is the maximal angular velocity of the motor
$Z=R+j\ x\ X,\ [\Omega]$	is the phase impedance (the sum of a phase resistance and phase reactance).

The inductor reactance X, [rad/sec] is:

$X = \omega\ x\ L = 2\pi\ x\ f\ x\ L$ (3)

where:

L, [H]	is the phase inductance
$\omega = 2 \text{ x } \pi x f$	f, [1/s] - is the frequency of the AC flowing through the circuit. f can be found from motor velocity as: $$f = \frac{V_{\max}}{2 \cdot Pitch}$$ for the linear motor, where *Pitch* is 180 electrical degrees. $$f = \frac{\omega_{\max}}{(NP/2)}$$ for the rotary motor where *NP* is the number of poles

The linear motor impedance is found from: (4)

$$|Z| = \sqrt{R^2 + (\omega \cdot L)^2} = \sqrt{R^2 + (2 \cdot \pi \cdot f \cdot L)^2} = \sqrt{R^2 + \left(\pi \cdot \frac{V_{\max}}{Pitch} \cdot L\right)^2}$$

The rotary impedance can be found from: (5)

$$|Z| = \sqrt{R^2 + (\omega \cdot L)^2} = \sqrt{R^2 + (2 \cdot \pi \cdot f \cdot L)^2} =$$
$$= \sqrt{R^2 + \left(2 \cdot \pi \cdot \frac{\omega_{\max}}{2 \cdot \pi \cdot (NP/2)} \cdot L\right)^2} = \sqrt{R^2 + \left(\frac{\omega_{\max}}{(NP/2)} \cdot L\right)^2}$$

Therefore, equation (1) can be represented as: (6)

$$I_{DB\_\max} = \frac{K_E \cdot V_{\max}}{\sqrt{3} \cdot \sqrt{R^2 + \left(\pi \cdot \frac{V_{\max}}{Pitch} \cdot L\right)^2}}$$

Equation (2) can be represented as: (7)

$$I_{DB\_\max} = \frac{K_E \cdot \omega_{\max}}{\sqrt{3} \cdot \sqrt{R^2 + \left(2 \cdot \pi \cdot \frac{\omega_{\max}}{(NP/2)} \cdot L\right)^2}}$$

In addition you should verify that the drive can withstand the heat dissipation.

Assuming 2V drop on each transistor, the maximum allowed power dissipation on one transistor is: *2 x I_{peak} x 1 sec.*

Thus, in order to protect the drive, you must verify that the following relation is complied with:

- In the case of a three-phase motor: $\frac{1}{2}\left(J\omega^2 + Mv^2\right) - \Delta P \le 6 \times I_{peak}$
- In the case of a DC motor: $\frac{1}{2}\left(J\omega^2 + Mv^2\right) - \Delta P \le 4 \times I_{peak}$

where J [kgm^2] is the total rotating inertia, M [kg] is the total moving mass, ω [rad/sec] is the angular velocity of the inertia and v [m/sec]is the linear velocity of the mass.

Both ω and v should be taken at the moment of activating the dynamic brake and are directly related to the user-determined **VELBRK** variable. Upon disable, dynamic braking starts only when the velocity is lower than this variable. ΔP is the power loss on the motor itself. It is recommended to disregard this element, thus ensuring that the total dissipated energy is lower than the allowed value of the drive.

16.2.2 Dissipation and Peak Current – Motor Verification

You must verify that the motor can withstand the dissipated heat and peak current. In some motors excessive current can lead to demagnetization.

16.2.3 Braking Torque Verification

You should also verify that the braking torque will not cause any damage to the mechanical system. Maximum braking torque is given by:

- $T_{\max} = K_T I_{\max} \frac{R}{Z} \approx K_T I_{\max}$ for a DC brushless motor
- $T_{\max} = K_T I_{\max}$, for a DC motor

where K_T is motor torque/ force constant.

16.2.4 Dynamic Brake - Verification Example

16.2.4.1 Linear Motor

A linear motor application with maximum speed of V_{max} = 1 [m/s] and 20A peak drive.

Three-phase motor specifications are listed in **Table 18**.

Table 18 Three-phase Motor Specifications - Linear Motor

Specifications	Unit	Value	Note
Total mass	Kg	4	
Motor force constant	n/Arms	17.5	
Motor peak current	Arms	18.4	
Back EMF	V/m/s	15	
Resistance per phase	Ω	1.2	
Induction per phase	mH	0.4	0.4 x 10^{-3} [H]
Magnetic pitch (180 electrical degrees)	mm	15	60.96 x 10^{-3}

Kinetic energy is $\frac{1}{2}\cdot 4\cdot 1^2 = 2$ Joule. This is lower than the permitte4d 6 x 20 = 120 J of the drive.

Impedance is $|Z| = \sqrt{R^2 + \left(\pi \cdot \frac{V_{\max}}{Pitch} \cdot L\right)^2} = \sqrt{1.2^2 + \left(\pi \cdot \frac{1}{15\cdot 10^{-3}} \cdot 0.4 \cdot 10^{-3}\right)^2} = 1.2$

Maximum current during braking process is $I_{DB\_\max} = \frac{K_E \cdot V_{\max}}{\sqrt{3}\cdot|Z|} = \frac{15\cdot 1}{\sqrt{3}\cdot 1.2} = 7.2$ [A]

This current is lower that the allowed 20A of the drive.

It is also lower that the 18.4A current of the motor, so the motor can easily withstand the current and resulting heat dissipation.

The braking torque will be: $\approx 17.5 \cdot \frac{7.2}{\sqrt{2}} = 89.1$ [N]

16.2.4.2 Rotary Motor

A rotary motor application with maximum speed of:

ω_{max} =500000 [count/s] = 500000/8000 x 2π [rad/s]=392.7 [rad/s] and 20A peak drive.

Three phase motor specifications are listed in **Table 19**.

Table 19 Three-phase Motor Specifications - Rotary Motor

Specifications	Unit	Value	Note
Rotor moment of inertia	Kg x m^2	0.000015	
Motor force constant	N x /Arms	0.14	
Motor peak current	Arms	22	
Back EMF	V/kRPM	16	
Resistance per phase	Ω	1.3	
Induction per phase	mH	2.1	4.1 x 10^{-3} [H]
Number of counts per rotation	counts	8000	
Number of poles		6	

Kinetic energy is $\frac{1}{2}\cdot 0.000015\cdot 392.7^2 = 1.15$ Joule. This is lower than the 6 x 20 = 120 J of the drive.

Impedance is $|Z| = \sqrt{R^2 + \left(\frac{\omega_{\max}}{(NP/2)}\cdot L\right)^2} = \sqrt{1.3^2 + \left(\frac{392.7}{6/2}\cdot 2.1\cdot 10^{-3}\right)^2} = 1.3\ \Omega$

Maximum current during the braking process is

$$I_{DB\_\max} = \frac{K_E\cdot\omega_{\max}}{\sqrt{3}\cdot|Z|} = \frac{16/(1000\cdot 2\cdot\pi)\cdot 60\cdot 392.7}{\sqrt{3}\cdot 2.7} = 12.8 \quad [A]$$

This current is lower than the allowed 20A of the drive.

It is also lower than the rated 22 A current of the motor, so the motor can easily withstand the current and the resulting heat dissipation.

The braking torque will be $\approx 0.14\cdot\frac{12.8}{\sqrt{2}} = 1.3$ [N x m]

Table 20 Motor Specifications

Specifications	Unit	Quantity
Peak current	Arms	18.4
Maximum continuous current - coils at 110° C	Arms	5.1
Maximum continuous power loss - all coils	W	115
Motor force constant	N/Arms	17.5
Back EMF phase-to-phase peak	V / m/s	15
Magnet pitch NM	mm	30
Resistance per phase	Ω	1.2

Table 20 Motor Specifications

Induction per phase	mH	0.4
Total mass	kg	4
Encoder resolution	lines/mm	250
Encoder multiplier		256

VELBRK should be: 1.5 x 1000 x 250 x 256 = 96 x 10^6 [counts/sec]

16.3 Dynamic Brake Operation Procedure

Caution	
	***Before operating the Dynamic Brake, verify that no damage will happen due to the dynamic Brake – see Section 16.2 - "Dynamic Brake - Safety Verification"***

1. Open **SPiiPlus MMI → Setup → Configurator**
2. Select the axis for dynamic braking.
3. Click **Motor Flags** and select **Dynamic Brake Mode** (bit 11).
4. Close the **Motor Flags** dialog box and click **Safety Parameters**.
5. Enter a value for the **Dynamic Brake Threshold Velocity (VELBRK) -** defines maximum feedback velocity to activate dynamic brake.

Note	
	*For safety reasons, the default value for* **VELBRK** *is zero for all axes.*

6. Dynamic braking will be activated automatically when the following two conditions are met:
   - **FVEL<axis>** is less than **VELBRK<axis>**
   - the axis drive is disabled

17 Appendix E: ASDF Measuring Motion Performance

17.1 Measuring Settling Time

1. Define the target position, see **Section 10.1 - "Criteria for Determining Whether Motor is in Position"**, by assigning values to **SETTLE** and **TARGRAD** variables.
2. Open **SPiiPlus MMI → Scope.** **Figure 54** depicts a Scope configured to measure settling time for a Nanomotion piezoceramic motor.
3. Assign a channel to monitor **RVEL** (reference velocity) for the axis. This shows when the reference velocity reaches zero.
4. Assign another channel to monitor **PE** (position error) for the axis.
5. Assign another channel to monitor bit 5 (**#MOVE**) of **AST** (axis state) for the axis. When motion has theoretically reached the target position, the bit will go low.
6. Assign another channel to monitor bit 4 (**#INPOS**) of **MST** (motion state) for the axis. When motion actually reaches the target position, the bit will go high.
7. Execute the motion.
8. Drag **Rider Cursor X1** to the point where **AST(*axis*)(5)** went low.
9. Drag **Rider Cursor X2** to the point where **MST(*axis*)(4)** went high.
10. The settling time = X2 – X1.

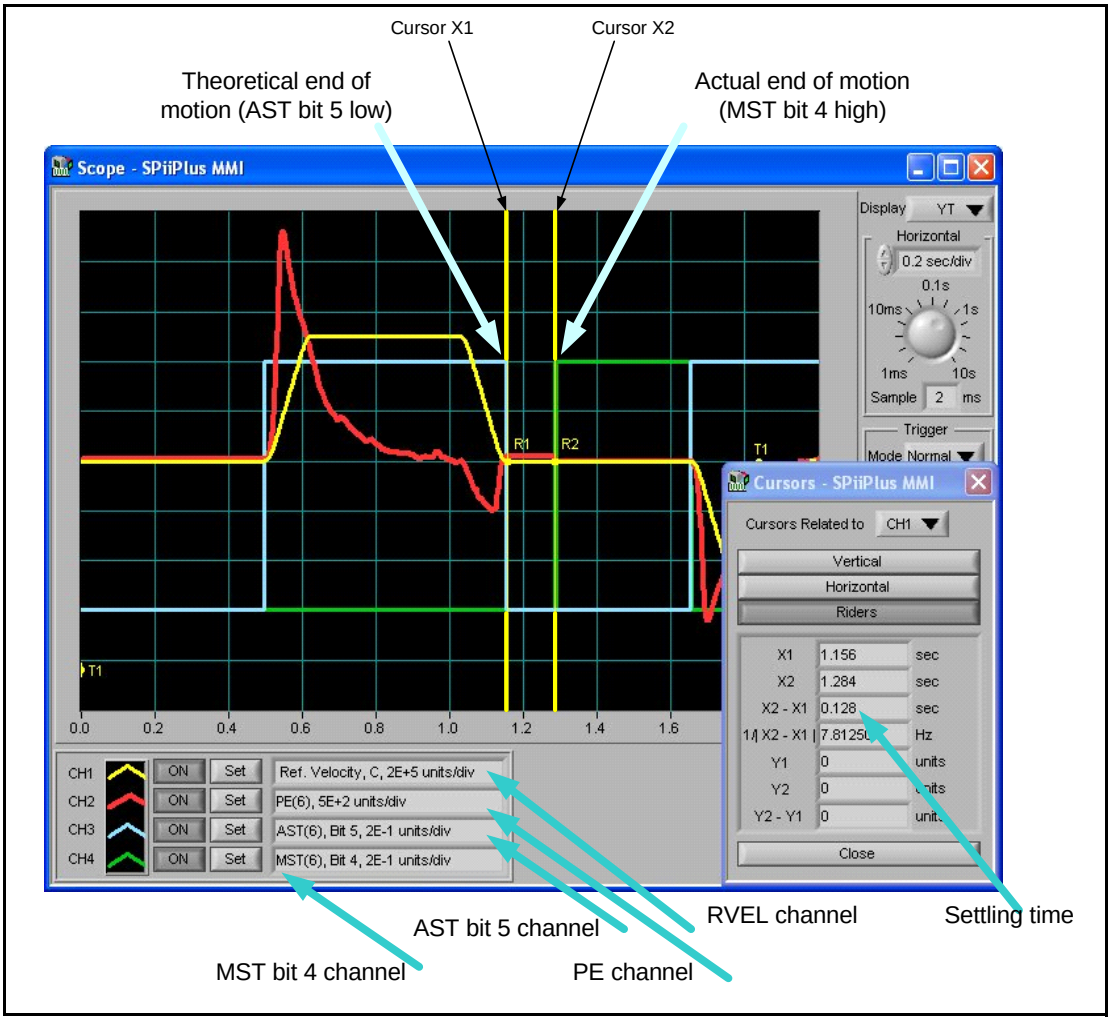
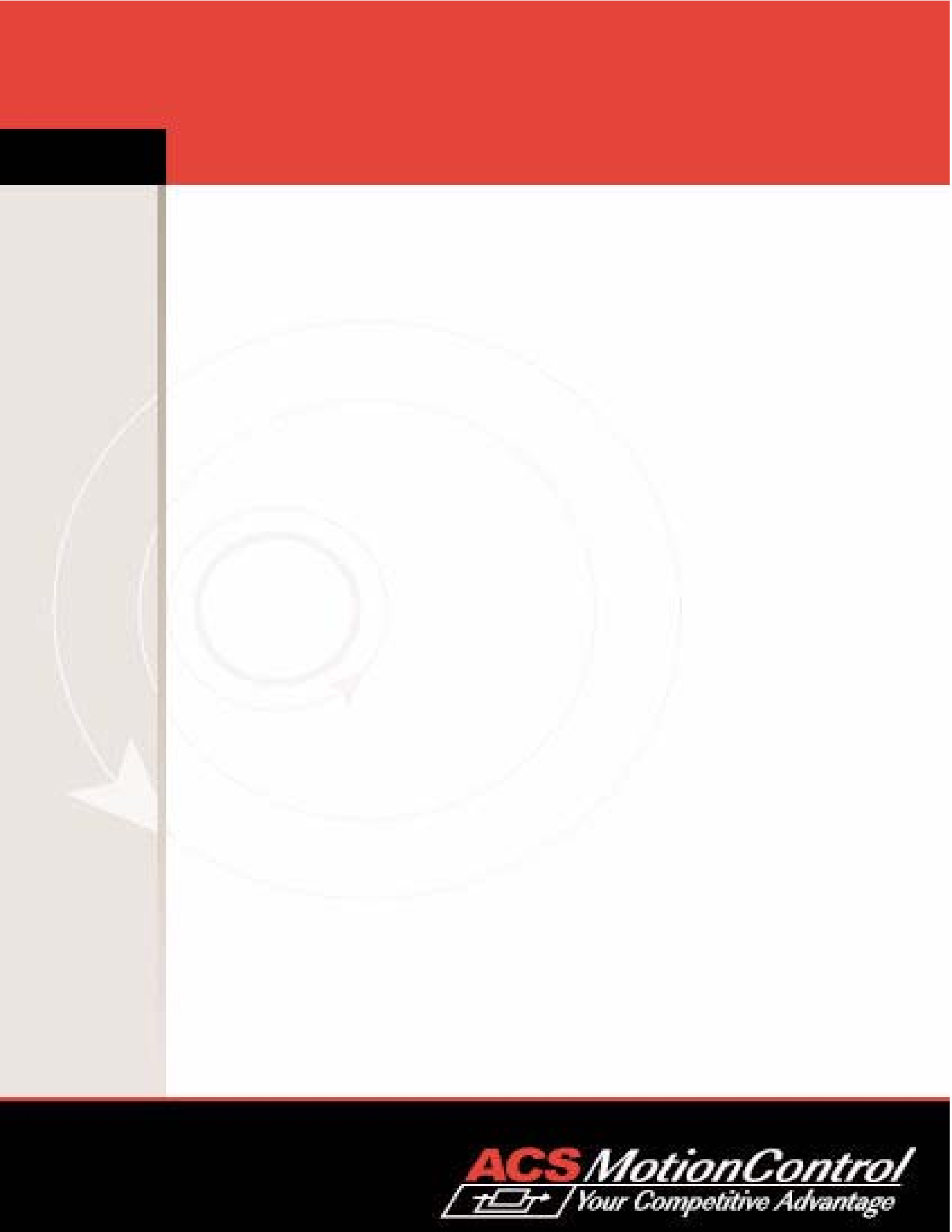


Figure 54 Measuring Settling Time (Piezoceramic Motor Example)


ACS MotionControl
Your Competitive Advantage