



SPiiPlus

Setup Guide

September 2020

Document Revision: 3.02

SPiiPlus

Release Date: September 2020

COPYRIGHT

© ACS Motion Control Ltd., 2020. All rights reserved.

Changes are periodically made to the information in this document. Changes are published as release notes and later incorporated into revisions of this document.

No part of this document may be reproduced in any form without prior written permission from ACS Motion Control.

TRADEMARKS

ACS Motion Control, SPiiPlus, PEG, MARK, ServoBoost, NetworkBoost and NanoPWN are trademarks of ACS Motion Control Ltd.

EtherCAT is registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany.

Any other companies and product names mentioned herein may be the trademarks of their respective owners.

www.acsmotioncontrol.com

support@acsmotioncontrol.com

sales@acsmotioncontrol.com

NOTICE

The information in this document is deemed to be correct at the time of publishing. ACS Motion Control reserves the right to change specifications without notice. ACS Motion Control is not responsible for incidental, consequential, or special damages of any kind in connection with using this document.

Revision History






Date	Revision	Description
September 2020	3.02	New Release
July 2020	3.01	ADK Update
March 2019	2.70	No system configuration wizard (Table 2-1, section 5) Icons System setup replaces EtherCAT/System setup in 2.7 DELK related parameters SAFIN, not SAFINI in 6.10.2
July 2018	2.60	Added homing methods Updated for Auto-mapping of IOMNT Added Current loop auto tuning Updated Open loop verification Added XRMST, XRMSM, XRMSTD, and XRMSTM
December 2017	2.50	Replaced EtherCAT Configurator with System Setup
June 2017	2.40	Reformatted with new template for SPiiPlus ADK Suite v2.40
October 2016	2.30.01	Added support for Absolute Encoders to SLPROUT , SLVROUT , SLCROUT
August 2016	2.30	Update for SPiiPlus ADK Suite v2.30 Reformatted using new template
May 2013	2.21	Absolute encoder setup procedure
December 2012	2.20	Update for 2.20
June 2012	2.15	Added appendix explaining dual loop control
November 2011	2.10	First release

Conventions Used in this Guide

Text Formats

Format	Description
Bold	Names of GUI objects or commands
BOLD + UPPERCASE	ACSPL+ variables and commands
Monospace + grey background	Code example
<i>Italic</i>	Names of other documents
Blue	Hyperlink
[]	In commands indicates optional item(s)
	In commands indicates either/or items

Flagged Text

	Note - includes additional information or programming tips.
	Caution - describes a condition that may result in damage to equipment.
	Warning - describes a condition that may result in serious bodily injury or death.
	Model - highlights a specification, procedure, condition, or statement that depends on the product model
	Advanced - indicates a topic for advanced users.

Related Documents

Documents listed in the following table provide additional information related to this document.

The most updated version of the documents can be downloaded by authorized users from www.acsmotioncontrol.com/downloads.

Document	Description
<i>SPiiPlus Command & Variable Reference Guide</i>	Describes all of the variables and commands available in the ACSPL+ programming language.
<i>SPiiPlus MMI Application Studio User Guide</i>	Explains how to use the SPiiPlus MMI Application Studio and associated monitoring tools.
<i>SPiiPlus Utilities User Guide</i>	Describes the use of SpiiPlus User Mode Driver, SPiiPlus Simulator, and UPGRADER.EXE.
<i>SPiiPlus C Library Reference</i>	C Methods, Properties, and Events for Communication with the Controller.
<i>SPiiPlus COM Library Reference</i>	COM Methods, Properties, and Events for Communication with the Controller.
<i>SPiiPlus .NET Library Reference</i>	.NET Methods, Properties, and Events for Communication with the Controller.

Table of Contents

1. Introduction	14
1.1 Scope of document	14
2. Getting Started	15
2.1 SPiiPlus Hardware Connection	15
2.2 SPiiPlus Software Installation	15
2.2.1 SPiiPlus Installation Disk	16
2.2.2 Software Installation	16
2.3 Safety Instructions	17
2.4 Product Licenses	17
2.4.1 About Licenses	17
2.4.2 License-Based Features Ordered with ACS EtherCAT Master	18
2.4.3 License-Based Features for ACS EtherCAT Slaves	19
2.4.4 Purchasing an ACS License	19
2.4.5 Entering New License String into the Controller	21
2.5 SPiiPlus Resources	21
2.5.1 Accessing SPiiPlus Programming Resources	21
2.5.2 Available SPiiPlus Resources	22
2.6 Activating SPiiPlus MMI Application Studio	22
2.7 Set Up New System	24
3. Establishing Communication with SPiiPlus MMI Application Studio	26
3.1 Setting Controller Properties	26
3.2 Connecting to the Controller	28
3.2.1 Serial Communication	28
3.2.1.1 Connecting via Serial Communication	28
3.2.1.2 Troubleshooting a Serial Connection	29
3.2.2 Ethernet Communication	29
3.2.2.1 Connecting via an Ethernet Network	30
3.2.2.2 Connecting via Ethernet Point-to-Point Communication	31
3.2.2.3 Troubleshooting Ethernet Network Connections	33
3.2.2.4 Troubleshooting Ethernet Point-to-Point Connection	34
3.3 Connecting via a Remote Host	34
3.4 Setting SPiiPlus Communication Parameters	35

3.5 The SPiiPlus Simulator	37
3.5.1 About the SPiiPlus Simulator	37
3.5.2 Connecting to the Simulator	38
4. System Setup	40
4.1 System Setup	40
4.1.1 Supported Network Masters	40
4.1.2 Supported Network Slaves	41
4.1.3 Supported Non-ACS Units	42
4.1.4 EtherCAT Cable Length	42
4.2 Numbering and Identification of Network Units, Axes and IOs	43
4.2.1 Unit ID Numbers	43
4.2.2 Axes Numbering in ACSPL+	43
4.2.3 I/O Numbering in ACSPL+	43
4.2.4 Logical and Physical Assignment of Network Units, Axes and IO Variables	44
4.3 ACSPL+ Programming in a Network Environment	44
4.4 Network Operation	45
4.4.1 Power-Up Sequence	45
4.4.2 Cable Disconnection	45
4.4.3 Losing 24V Power	45
4.4.3.1 Network Monitoring	45
4.5 System Setup	46
4.5.1 Configuring the Network	46
4.5.1.1 Automatic System Setup	46
5. System Hardware Configuration	48
6. Axis Setup and Tuning	49
6.1 Principles of Motion Control	49
6.1.1 Control Loops	49
6.1.2 SPiiPlus Motion Control Algorithms	50
6.1.2.1 PI Filter	50
6.1.2.2 Low Pass Filter	51
6.1.2.3 Notch Filter	52
6.1.2.4 BI-Quad Filter	52
6.2 About the Adjuster Wizard	56
6.2.1 Working in Adjuster Wizard Task Windows	57

6.2.1.1 Data Action Buttons	57
6.2.1.2 Adding Data to Component Database	58
6.2.1.3 Incompatibility Symbol	58
6.3 Starting Adjuster Wizard	59
6.4 Selecting Axis Setup Initial Values	60
6.5 Initialization	62
6.6 Axis Architecture	62
6.6.1 User Units	63
6.6.2 User Defined User Unit	64
6.7 Components	65
6.7.1 Motor	65
6.7.2 Drive	66
6.7.3 Feedback	66
6.7.3.1 Absolute Encoder Setup Procedure	67
6.7.4 Calculate Parameters	68
6.8 Safety and Protection	69
6.8.1 Motion Parameter Limits	70
6.8.2 Current Limits	71
6.8.3 Position Errors	73
6.8.4 Position Limits	75
6.9 Miscellaneous Definitions	75
6.9.1 Motion Completion	75
6.9.2 Enable/Disable/Brake	76
6.9.3 Dynamic Brake	78
6.9.4 Home Switch	79
6.10 Verification	79
6.10.1 Feedback	80
6.10.2 Switches	82
6.10.3 Stop, Alarm and Brake	84
6.11 Axis Setup and Tuning	85
7. Current Loop and Current Phase Offset Setup	86
7.1 Current Loop	86
7.2 Current Phase Offset	91
8. Commutation	94

8.1 Adjuster and Commutation	94
8.1.1 Commutation Overview	94
8.1.2 Commutation Technical Details	94
8.1.3 The Adjuster Commutation Task	95
8.2 Performing the Commutation Task	96
8.2.1 Preferences	97
8.2.2 Advanced Parameters	99
8.2.3 Commutation Startup Program	100
8.2.4 Commutation Program Output Panel	102
8.3 Troubleshooting Commutation	103
9. Open Loop and Position Verification	104
9.1 Open Loop Verification	104
9.2 Position Verification	106
10. Position and Velocity Loops	109
10.1 Velocity Loop	110
10.2 Position Loop	113
11. Save to Flash	117
12. Sin-Cos Encoder Compensation	118
12.1 Activating Sin Cos Encoder Compensation	118
12.2 Sin Cos Encoder Compensation Window	119
12.2.1 Measurement Curve Display	120
12.2.2 Compensation Execution Panel	120
12.2.3 Parameters Panel	122
12.2.3.1 Advanced Tab	122
12.2.3.2 Cursors Tab	124
12.2.3.3 Statistics Tab	125
12.2.4 Running Sin Cos Encoder Compensation	126
13. Homing Programs	128
13.1 Homing Program Using Limit Switches	128
13.2 Homing Program Using Hard Stops	129
14. Protecting Application	131
14.1 Starting Protection Wizard	131
14.2 Define Protection	132
14.2.1 Setting Application Protection	133

14.2.2 Setting Intellectual Property Protection	136
14.2.3 Setting Custom Protection	136
14.2.3.1 Custom buffer protection	136
14.2.3.2 Custom variable protection	137
14.2.3.3 Advanced protection options	138
14.2.4 Update Protection	138
14.2.5 Remove Protection	140
14.2.6 View Protection	143
14.2.7 Entering Incorrect Password	145
Appendix A. Working with a Gantry Axis	147
A.1 Overview	147
A.2 Gantry Implementation Using Connect And Depends Commands	147
A.3 Gantry Implementation	148
A.3.1 Gantry Setup and Configuration	149
A.3.2 Commutation and Homing	150
A.3.3 Gantry Operation	151
Appendix B. Dual Loop Control	153
B.1 Dual Loop Basics	153
B.2 ACSPL+ Variables	155
B.2.1 MFLAGS	155
B.2.2 SLVRAT	155
B.2.3 SLVKP	155
B.2.4 XVEL	155
B.2.5 EFAC	155
B.2.6 FACC	155
B.2.7 Routing Variables	155
B.2.7.1 SLPROUT	156
B.2.7.2 SLVROUT	157
B.2.7.3 SLCROUT	158
B.2.8 Configuring Dual Loop Control Procedure	159
B.2.8.1 AXIS Setup	159
B.2.8.2 LOAD Setup	160
Appendix C. Working with Dynamic Braking	164
C.1 About Dynamic Braking	164

C.2 Dynamic Brake - Safety Verification	164
C.2.1 Dissipation And Peak Current – Drive Verification	164
C.2.2 Dissipation and Peak Current – Motor Verification	167
C.2.3 Braking Torque Verification	167
C.2.4 Dynamic Brake - Verification Example	167
C.2.4.1 Linear Motor	167
C.2.4.2 Rotary Motor	168
C.2.5 Dynamic Brake Operation Procedure	169
Appendix D. ASDF Measuring Motion Performance	172
D.1 Measuring Settling Time	172

List Of Figures

Figure 2-1. SPiiPlusMMI Application Studio Startup Window	23
Figure 3-1. SPiiPlus Controller Properties Window	27
Figure 3-2. RS-232 Serial Connection	28
Figure 3-3. Types of Ethernet Connection	30
Figure 5-1. Motion Control System Schematic	49
Figure 5-2. Control Loops	50
Figure 5-3. PI Filter Schematic	51
Figure 5-4. PI Filter Frequency Response	51
Figure 5-5. Bi-Quad Configured as a Notch Filter	53
Figure 5-6. Bi-Quad Configured as a 2nd Order Lead Filter	54
Figure 5-7. Bi-Quad Configured as a 2nd Order Lag Filter	55
Figure 5-8. Bi-Quad Configured as a 2nd Order Low Pass Filter	56
Figure 7-1. Commutation Options	101
Figure 9-1. Velocity and Position Loop Adjustment - General Layout	110
Figure 9-2. Velocity Loop Motion Manager Panel	111
Figure 9-3. Example Scope Display of Well Adjusted Velocity Loop	113
Figure 9-4. Position Loop Manager Panel	114
Figure 9-5. Example Scope Display of Well Adjusted Position Loop	116
Figure 12-1. Homing	128
Figure 14-1. Typical Gantry	147
Figure 14-2. Gantry Axis Profile Generation	148
Figure 15-1. Dual Loop System	153
Figure 15-2. Controller in Dual Loop Block Diagram	154
Figure 17-1. Dynamic Brake Implementation by the Digital Drive	164
Figure 18-1. Measuring Settling Time (Piezoceramic Motor Example)	172

List of Tables

Table 2-1. Tool Use Requirement	18
Table 2-2. ACS EtherCAT Master License-Based Features	18
Table 5-1. Digital Outputs for Brake Control in SPiiPlus	77
Table 5-2. Digital Outputs for Brake Control in MC4U	78
Table 16-1. SLPROUT Values	156
Table 16-2. SLVROUT Values	157
Table 16-3. SLCROUT Values	158
Table 17-1. Three-Phase Motor Specifications - Linear Motor	167
Table 17-2. Three-Phase Motor Specifications - Rotary Motor	168

1. Introduction

1.1 Scope of document

This guide provides guidance in how to configure and adjust SPiiPlus systems to work with supported types of motors and feedback devices. The guide applies to SPiiPlus stand-alone motion controllers and modules as well as MC4U Control Modules with NT motion controllers and modules installed.

The principle application employed for setting up SPiiPlus systems is the **SPiiPlus MMI Application Studio** which is supplied with the CD package accompanying all SPiiPlus products.

Setting up the SPiiPlus system basically involves:

- > Establishing communication between the SPiiPlus MMI Application Studio and the controller.
- > Where an EtherCAT network is involved, the SPiiPlus MMI Application Studio System Setup is employed to configure the network.
- > If an MC4U Control Module is employed, the SPiiPlus MMI Application Studio System Configuration Wizard is used for establishing its configuration.
- > After the above have been accomplished, the SPiiPlus MMI Application Studio Adjuster Wizard tool is employed to establish the nature of the control loops. Various types of control loops and how they are set up are given.

2. Getting Started

This chapter provides the basics for setting up a SPiiPlus system.

2.1 SPiiPlus Hardware Connection

Installation and the electrical interface information for specific motion controllers is provided in the hardware guide associated with the motion controller 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



When an MC4U unit is not supplied bus voltage, a component failure fault will be reported. The fault is system wide and will prevent all axes from operating unless the fault is masked or bus voltage is supplied to the power supply. When a component failure is reported, the affected power supply is identified by its I2C address, access the System Information Viewer (see *SPiiPlus MMI Application Studio User Guide*) to determine the faulty unit.

2.2 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 in the following Microsoft® Windows® environments:

- > Windows® Vista SP1 and later (32-bit and 64-bit)
- > Windows® 7 (32-bit and 64-bit)
- > Windows® 8 (32-bit and 64-bit)
- > Windows® 10 (32-bit and 64-bit)
- > Windows® Server 2003 (32-bit and 64-bit)
- > Windows® Server 2008 (32-bit and 64-bit)

> Windows® Server 2008 R2 (64-bit)



Windows® XP SP2 (64-bit) and Windows® XP SP3 (32-bit) are no longer supported.

Each new version of SPiiPlus software is installed in a new folder (with the new version name) under **C:\Program Files\ACS Motion Control\SPiiPlus ADK Suite vx.xx**



For SPiiPlus system software previous to v2.30, the file directory is under **C:\Program Files\ACS Motion Control\SPiiPlusNT Suite x.xx**.

2.2.1 SPiiPlus Installation Disk

The SPiiPlus ADK Suite 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 Application Studio – Windows-based software tool for configuring, adjusting, programming and monitoring the motion.

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

SPiiPlus Utilities - The SPiiPlus User Mode Driver, which provides the communication link between the host computer and the motion controller.

SPiiPlus Documentation – Data sheets and user guides documentation for the SPiiPlus products. All the documents are in PDF format.

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

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



ACS Motion Control recommends using the most updated SPiiPlus tools and documentation.

2.2.2 Software Installation

To install the SPiiPlus software package:

1. Insert the **SPiiPlus ADK Suite** installation CD into the PC.
2. Follow the on-screen installation instructions.

2.3 Safety Instructions



The safety procedures outlined in this section, along with safety instruction detailed in the documentation of each network element, must be carefully observed in order to achieve safe and optimal operation of ACS networking provisions.

Make sure that the following guidelines are addressed and observed prior to powering or handling any of the network elements:

- > Installation and maintenance must be performed by qualified personnel only. Such a person must be trained and certified to install and maintain high power electrical equipment, servo systems, power conversion equipment and distributed networks.
- > Prior to powering up the system, ensure that all network components are connected to earth grounding.
- > Keep all equipment covers close and secured during all time.
- > Maintenance should be performed only after the relevant network element has been powered down, and all associated and surrounding moving parts have settled in their safe mode of operation. Certain drives require longer times in order to fully discharge. Follow the hardware guide of each element and observe the residual discharge time specified.
- > Avoid contact with electrostatic-sensitive components and take the required precautions.

2.4 Product Licenses

2.4.1 About Licenses

Licenses (sent as software strings) are keys used to set certain general properties of the network, such as the required number of axes, special features and type and number of non-ACS network elements.



The license string allows controller options only for the specific controller and cannot be used for another controller.

The network units are initially ordered and delivered with a pre-defined configuration and feature set. This set consists of product-specific features, such as, number of axes, number of Sin-Cos encoders, hardware configuration parameters. These features are stored in the products themselves. Other network parameters, such as special servo features and number of total axes, and number of non-ACS IO devices, are stored in the master controller. For example, a UDMnt-2 can be ordered with none, with 1 or 2 Sin-Cos multiplies, in one or two axes configuration. On the network level, the master controller can be ordered with or without ServoBoost™ as a special servo feature.

The ordered master controller is factory pre-programmed to the maximum number of axes, number of non-ACS servo axes, non-ACS stepper axes and non-ACS IO nodes. ServoBoost™ and other customized servo algorithms are ordered at the Master/Network level as well.

The number of network elements (units and EtherCAT nodes) is limited up to the maximal number listed defined in [Table 2-2](#). The limitation may result from the specific master chosen, by the resulting number of supported nodes, and the required axes.

In order to change a network's configuration, that relates to network-level features, after the initial order, the user may have to obtain a specific software license. Changing network configuration (addition or subtraction of units, adding, removing or replacing modules within the MC4U, or adding programmable features) may require modification of the network configuration in addition to the above described software license., as outlined in [Table 2-1](#).

Table 2-1. Tool Use Requirement

Action	System Setup	Obtain License
Changing order of units (cabling only) within a given network	Yes	No
Changing DIP switch number	Yes	No
Addition of units	Yes	Yes (only in case limits are exceeded)
Removal of units	Yes	No
Modifying MC4U configuration	No	See Note
Adding axes (MC4U, ACS units or non-ACS)	Yes	Yes
Adding non-ACS IO units	Yes	Yes



A license is needed when a network modification (addition of a unit or units, or MC4U axes by addition of modules) increases the total number of network axes beyond the ordered axes (defined by the master) and specified minimal **CTIME** and when it increases the total license-based master or slave features.

2.4.2 License-Based Features Ordered with ACS EtherCAT Master

The following features are ordered, pre-programmed and delivered in any ACS EtherCAT Master.

Table 2-2. ACS EtherCAT Master License-Based Features

Requirement	Description	Factory Setting
Total number of axes	All servo axes in the network (4, 8, 16, 32, or 64). Two performance levels of MPUs within NTM controllers are supported, as function of needed axes and CTIME.	As ordered
Input Shaping	Optional algorithm.	As ordered

Requirement	Description	Factory Setting
Customized servo algorithms	Support of customized servo features, such as ServoBoost™.	As ordered
Number of non-ACS servo axes	Servo and stepper axes provided by (certified) non-ACS EtherCAT drives. Refer to ACS price list for detailed upgrade information	As ordered
Number of non-ACS stepper axes	Axes provided by (certified) non-ACS EtherCAT drives. Refer to ACS price list for detailed upgrade information	As ordered
Number of non-ACS I/O nodes	IOs provided by (certified) non-ACS EtherCAT devices. Refer to ACS price list for detailed upgrade information	As ordered

In order to change any of these parameters after the network has been delivered, the user has to obtain a new license from ACS.

2.4.3 License-Based Features for ACS EtherCAT Slaves

ACS EtherCAT Slave units are delivered according to the ordered configuration, which addresses the product type, number of axes, hardware configuration, and number of Sin-Cos encoders.



Once delivered, however, no changes to the ordered configuration can be made in the field. If changes are required, the product has to be returned to ACS to effect the change.

2.4.4 Purchasing an ACS License

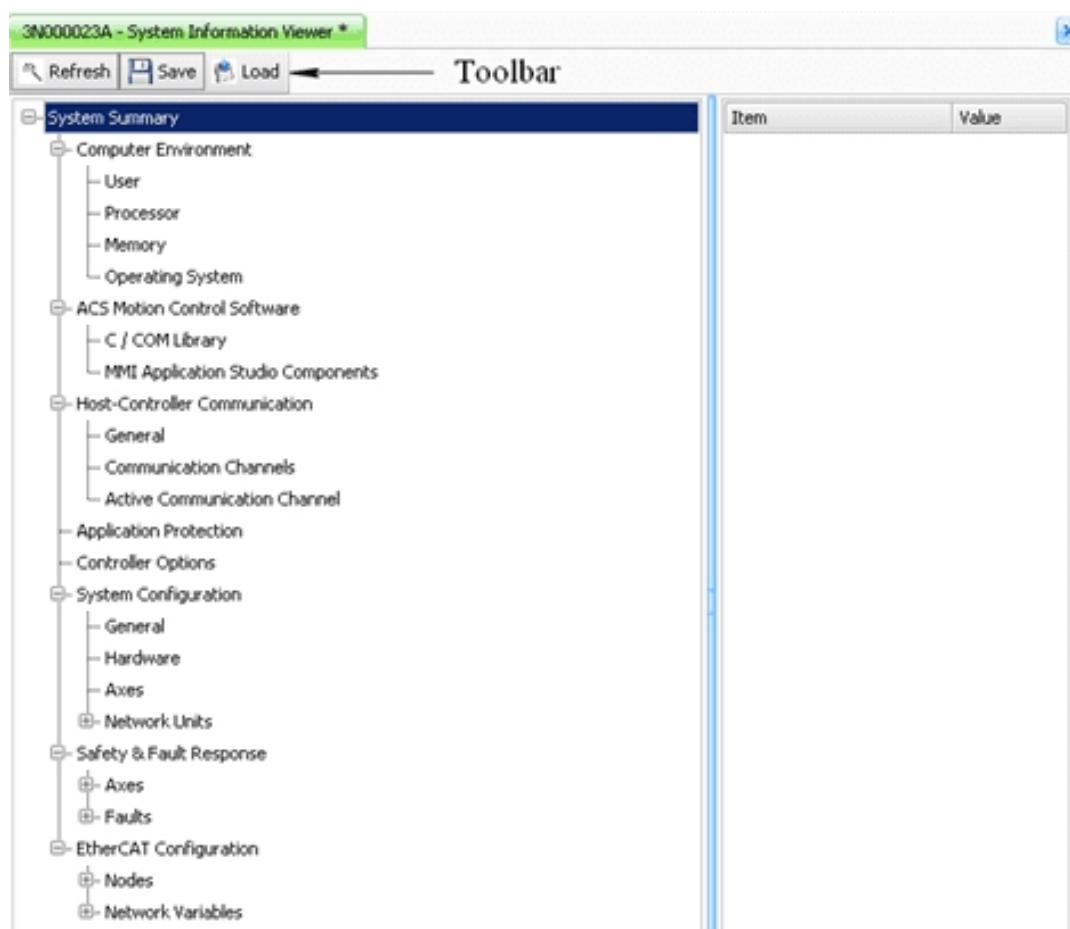
In order to increase, or change, any of the capabilities listed in above, the user has to obtain a license from ACS Motion Control. The license is purchased from ACS as a product.

In order to obtain a new license, the user has to provide the following information to ACS:

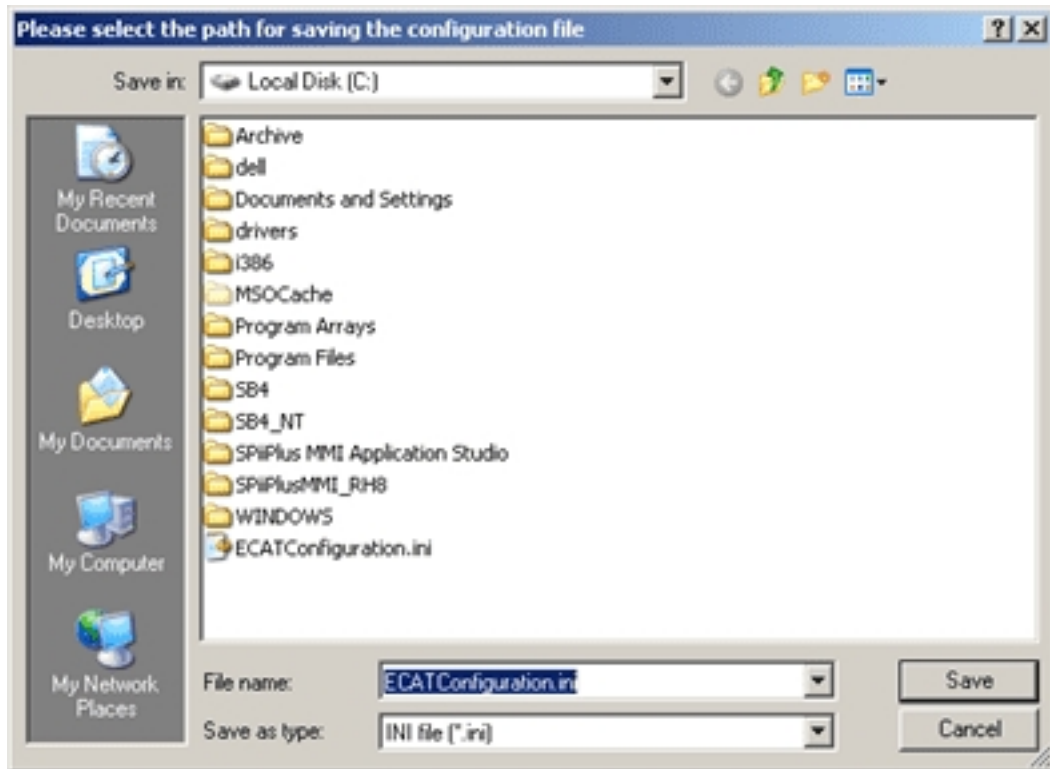
- > The report generated by the SPiiPlus MMI Application **Studio System Information Viewer**.
- > The requested features

To generate the System Information Report:

1. In the SPiiPlus MMI Application Studio select **Toolbox > Utilities > System Information Viewer**. The System Information window listing the data for all components is displayed:



2. Click Save () on the toolbar. The browser is displayed.



3. Browse to the location where you want to save the information and click **Save**.

The data is saved in a file with an **.ini** extension. It is this file that you send to ACS Motion Control when purchasing a new license.

2.4.5 Entering New License String into the Controller

The new license string is entered into the controller by copying the following command:
#SCO "license string" (that is sent from ACS Motion Control) into the SPiiPlus MMI Application Studio **Communication Terminal** and executing it.

After entering the license string, the controller has to be restarted.

2.5 SPiiPlus Resources

2.5.1 Accessing SPiiPlus Programming Resources

The SPiiPlus program group can be accessed from Windows: **Start > Programs > SPiiPlus NT Suite**, for example:



2.5.2 Available SPiiPlus Resources

Items that you can access in this manner are:

- > SPiiPlus product data sheets in PDF format
- > SPiiPlus training documents in PowerPoint format
- > SPiiPlus product user guides
- > SPiiPlus MMI Application Studio
- > SPiiPlus MMI Application Studio Help file
- > SPiiPlus User Mode Driver




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.

2.6 Activating SPiiPlus MMI Application Studio

SPiiPlus MMI Application Studio can be activated either from the Windows **Start** menu:



Or by double-clicking  on the Windows Desktop.
The SPiiPlus MMI Application Studio startup window is displayed.

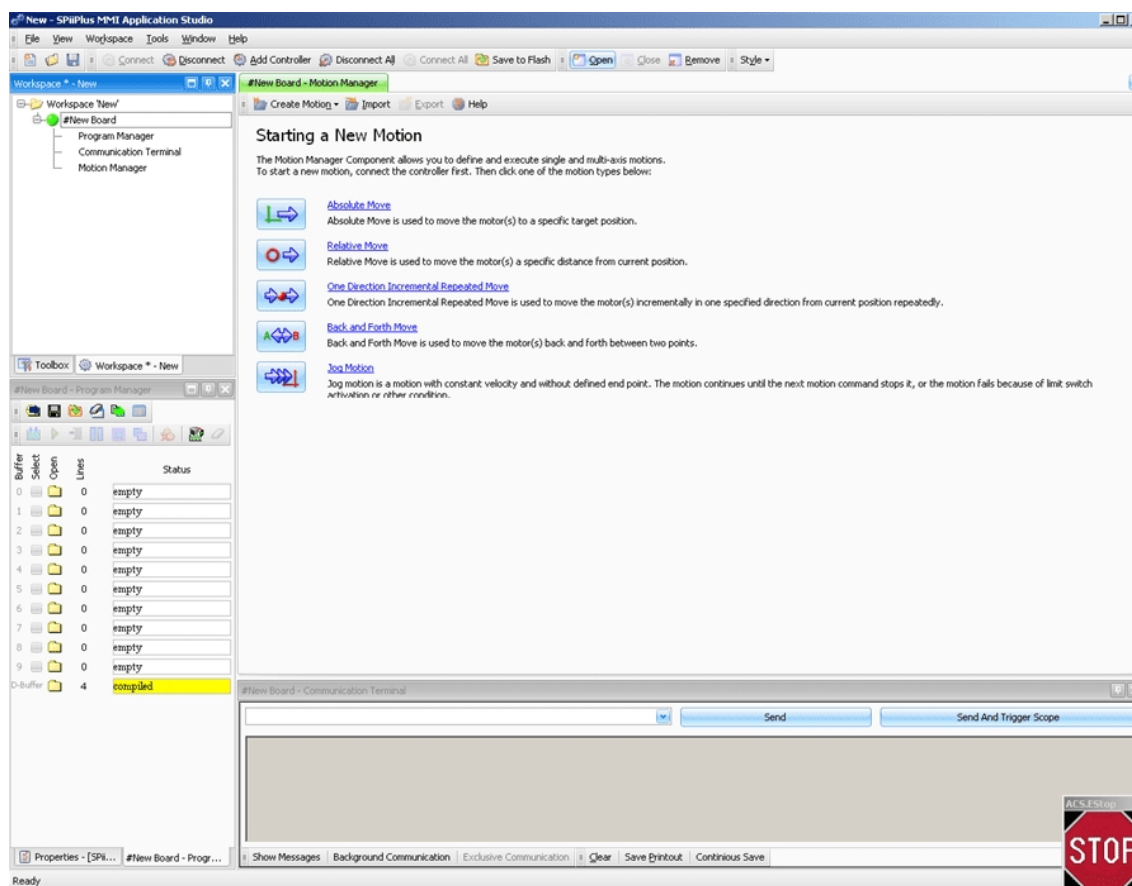


Figure 2-1. SPiiPlusMMI Application Studio Startup Window

For a complete description of the elements making up the SPiiPlus MMI Application Studio Startup window, see *SPiiPlus MMI Application Studio User Guide*.

2.7 Set Up New System

The steps to be followed when setting up your new ACS Motion Control system are:

1. Establishing Communication with SPiiPlus MMI Application Studio

You first have to establish communication with the controller in order to set all the parameters in the steps that follow.

2. System Setup

3. System Setup

SPiiPlus technology enables you to link several devices (ACS Motion Control devices as well as certain approved non-ACS devices) together in a master/slave configuration. Once all the devices have been physically connected, you have to perform this step to establish the parameters of the network.

4. System Hardware Configuration

In this step you use the **System Setup Component** to scan your system in order to define the physical parameters of each device. This step involves: defining the configuration in the database, verifying that the configuration is correct, and then loading the configuration data into the master controller. Once this step is complete, the master controller knows what type of devices are in the system and what they are controlling.

The physical parameters include:

- > Whether the device is a motion controller or I/O device
- > Whether the controller is a master or slave
- > The motherboard being used (for MC4U units)
- > Power supply that is installed (for MC4U units)
- > Drivers that are installed (for MC4U units)
- > Axes to which the devices are connected
- > I/O connections

5.

In this step you begin by defining the motion parameters for each axis through the **Adjuster Wizard**. Then you proceed to adjust the motion parameters based on the nature of the axis parameters as well as the motors that are being driven. The adjustment of the specific types of motion parameters is covered in [Current Loop and Current Phase Offset Setup](#), [Commutation](#), [Open Loop and Position Verification](#), [Position and Velocity Loops](#), and [Save to Flash](#).

6. Protecting Application

At this point, the controller has been in the configuration mode, meaning that it can be written to. Changing the controller mode to **Protected** limits changes that can be made to controller variables and program buffers. The 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.

Further specialized motion application considerations are covered in the appendices.

3. Establishing Communication with SPiiPlus MMI Application Studio

This is the first operation that must be performed before you can start setting up your system. There are three types of communication:

- > RS-232 serial connection
- > Ethernet connection which can be either point-to-point (direct to controller) or over a network
- > Remote connection through a host connected to the motion controller

This chapter also provides details for connecting to the SPiiPlus Simulator utility.

The steps in establishing communication are:

1. Set controller properties
2. Connect to the controller



If you have more than one SPiiPlus Motion Controller in your system, use the SPiiPlus MMI Application Studio **Add Controller** function and repeat the communication setup for each motion controller. See *SPiiPlus MMI Application Studio User Guide* for details on this function.

Communication setup for each type of channel is described here using the SPiiPlus MMI Application Studio SPiiPlus Controller Properties dialog window through which you configure the controller's communication settings.



Certain parameters, such as logging and remote connections, can also be set using SPiiPlus User Mode Driver - see *SPiiPlus Utilities User Guide*.

3.1 Setting Controller Properties

To set controller properties - double-click the controller in the Workspace Tree, as shown:

3. Establishing Communication with SPiiPlus MMI Application Studio



The SPiiPlus Controller Properties window is displayed:



Figure 3-1. SPiiPlus Controller Properties Window

1. In the **General** tab enter an identifying name in the **Controller Name** field.
2. Enter the connection timeout value, in milliseconds, in the **Connection Timeout** field.

3.2 Connecting to the Controller

3.2.1 Serial Communication



The SPiiPlus MMI Application Studio serial communication is not intended to be used for safety purposes.

All SPiiPlus motion control products include two RS-232 serial communication channels (COM1 and COM2). The default settings of RS-232 are: 8 data bits, no parity and a regular stop bit.

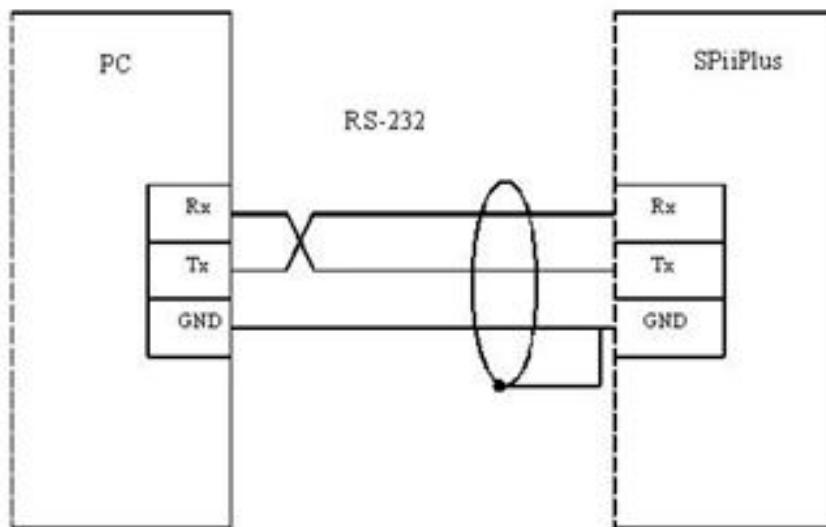



Figure 3-2. RS-232 Serial Connection

3.2.1.1 Connecting via Serial Communication

To connect the host computer to the SPiiPlus Motion Controller by way of the serial port:

1. In the **General** tab select **Serial**.



2. Select the communication port from the **Port** dropdown list.
3. Select the baud rate from the **Rate** dropdown list.
4. Click **Connect** ().

3.2.1.2 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.

3.2.2 Ethernet Communication

Two types of Ethernet connections are supported:

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

These two types of Ethernet connections are illustrated in [Figure 3-3](#).

3. Establishing Communication with SPiiPlus MMI Application Studio

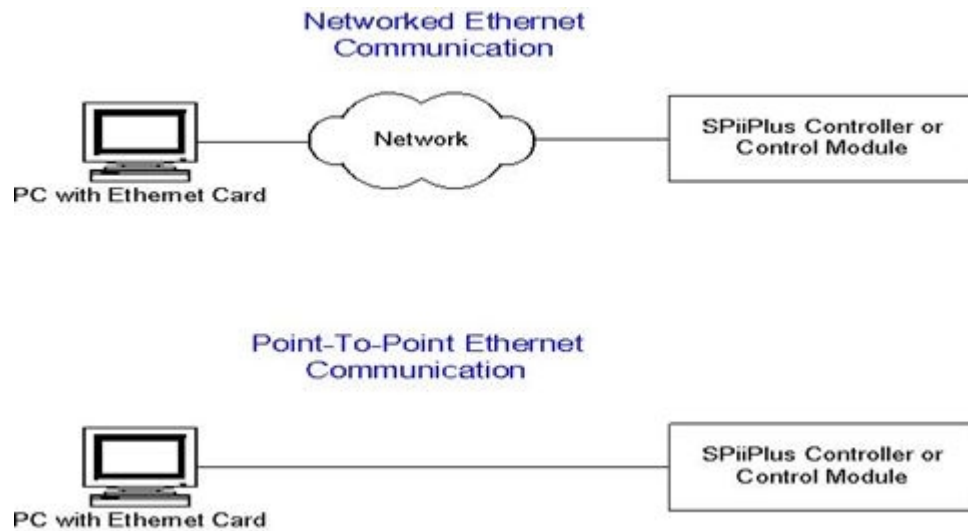


Figure 3-3. Types of Ethernet Connection

3.2.2.1 Connecting via an Ethernet Network

To connect the host computer to the motion controller via an Ethernet network:

1. In the **General** tab select **Ethernet**.



2. Select the IP Address from the **Controller IP Address** dropdown list.



The factory default TCP/IP address is 10.0.0.100.

3. Select **Network** from the Network Type dropdown list.



4. Click **Connect** ().

3.2.2.2 Connecting via 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.



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

To connect the host computer to the motion controller via point-to-point Ethernet:

1. In the **General** tab select **Ethernet**.

3. Establishing Communication with SPiiPlus MMI Application Studio



2. Select the **IP** Address from the **Controller IP Address** dropdown list.



The factory default TCP/IP address is 10.0.0.100.

3. Select **Point to Point** from the Network Type dropdown list.



4. Click **Connect** ().

3.2.2.3 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.

If two controllers have the same IP, an error message (code 165) appears. Later, to connect via Ethernet network to the controller with a unique IP, you should reboot the controller by right-clicking the controller in the Workspace Tree and selecting

Controller Reboot:



3.2.2.4 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 is selected in the SPiiPlus MMI Application Studio.

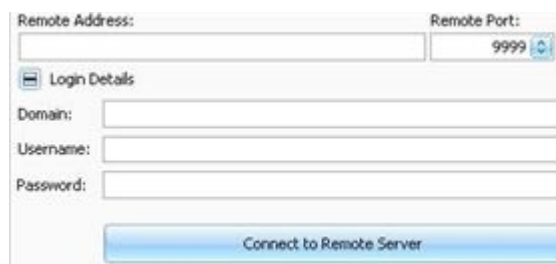
3.3 Connecting via a Remote Host

To connect to the motion controller through a remote host:

1. In the **General** tab select the **Remote Connection** checkbox.



2. Enter the IP of the remote computer in the **Remote Address** field.
3. Enter the port number of the remote computer in the **Remote Port** field.
4. If the remote computer has login requirements, click **Login Details**:



Enter the **Domain** name, **Username** and **Password**.

5. Click **Connect to Remote Server** ().

3.4 Setting SPiiPlus Communication Parameters

Once you have established a communication connection with your SPiiPlus Motion Controller, you have the option of setting certain communication parameters and store them into the controller's flash memory

1. In the SPiiPlus Controller Properties window, click the **Communication Parameters** tab:

3. Establishing Communication with SPiiPlus MMI Application Studio



This tab enables you to set the following communication parameters:

- > COMMFL
- > DISPCH
- > BAUD
- > TCPIP
- > TCPIP2
- > TCPPOINT
- > UDPPORT

For details on these options see the *SPiiPlus MMI Application Studio User Guide*.

2. Once you have set the parameters, click **Save to Flash** (). The Save to Flash window is displayed:

3. Enter your user name in the **User** field.
4. Enter the application name in the **Application** field.

You can, if you desire, enter free text remarks in the **Remarks** field.



By default all data is selected. You have the option of selecting the data you want to load into the controller by clicking (thereby deselecting) the checkbox of the data you do not want to be saved.

5. Click **Save**.

3.5 The SPiiPlus Simulator

This section describes using the SPiiPlus Simulator. For complete details of the Simulator see the *SPiiPlus Utilities User Guide*.

3.5.1 About the SPiiPlus Simulator

The SPiiPlus Simulator allows you to work with the SPiiPlus MMI Application Studio, SPiiPlus C Library or SPiiPlus COM Library without being physically connected to a controller, drives, or motors. The Simulator thus is a powerful tool for use during application development and debugging before connecting to actual hardware.

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, however **TIME=1** is not equal to 1 millisecond as when working with a real SPiiPlus.
- > Some hardware-based programming features, including: Index, MARK, PEG, and analog I/O (**AIN** and **AOUT**).
- > Some SPiiPlus MMI Application Studio **Communication Terminal** commands: **#HWRES**, **#RESET**, **#U**, **#IR**, **#SI**, **#SIR**.
- > The **write/read** commands.
- > The SPiiPlus MMI Application Studio **Application Saver** and **Application Loader**.



The Simulator does, however, provide limited support for SPiiPlus MMI Application Studio **Adjuster Wizard**.

3.5.2 Connecting to the Simulator

This section provides the procedure for connecting to the SPiiPlus Simulator via the SPiiPlus MMI Application Studio, for other methods see the *SPiiPlus Utilities User Guide*.

To connect to the Simulator:

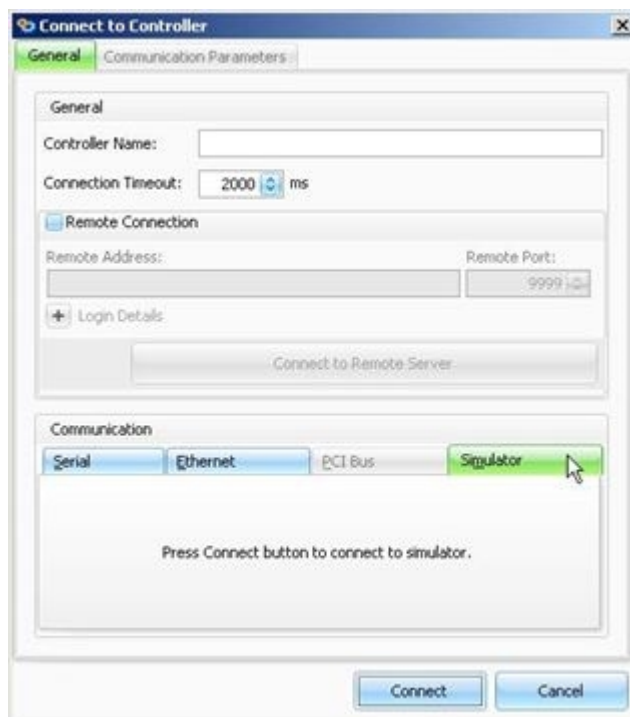
1. Click **Workspace** on the SPiiPlus MMI Application Studio Menu bar, and select **Add Controller**.



The Connect to Controller dialog is displayed.

2. Click **Simulator**.

3. Establishing Communication with SPiiPlus MMI Application Studio



3. Click **Connect** (). The “controller” is added to the Workspace Tree.

4. System Setup

4.1 System Setup

The system setup process includes setting up the EtherCAT network and configuring the units connected to the network.

ACS network solutions are based on a master device and multiple network elements, connected over an EtherCAT cable. The combination of EtherCAT technology and ACS network capabilities ensure that the master and all network elements are tightly time-synchronized.

An ACS EtherCAT network consists of a master (see [Supported Network Masters](#)) that is always positioned as the first network element. The subsequent elements can be any combination of ACS and non-ACS EtherCAT network elements (see [Supported Network Slaves](#)). A daisy chain cabling is used to connect the master to the network elements: The master is connected by a single EtherCAT cable to the first "slave", which connects to the second "slave", which in turn connects to the third one. This process is repeated until the last "slave" element is reached. The EtherCAT cables support bi-directional, full duplex communication.



No redundancy cable, or ring topology, is supported.

The number of supported elements is determined by the type of master used, and the software version installed. The axes and IO numbering is determined through the System Setup Wizard, a component of the SPiiPlus MMI Application Studio, as detailed in the following sections and by setting the front panel DIP switches on ACS network elements that contain them.

4.1.1 Supported Network Masters

The following ACS Motion Control units can serve as a network master:

- > SPiiPlusNTM Network Manager

SPiiPlusNTM-32 and SPiiPlusNTM-64 are stand-alone master controllers which differ by the maximal number axes they support. The controllers support both ACS and non-ACS network elements in various combinations. The SPiiPlusNTM-32 supports up to 32 axes and 64 nodes. The SPiiPlusNTM-64 supports up to 64 axes and 64 nodes.

- > SPiiPlusNT motion controllers

The SPiiPlusNT product line consists of:

- > SPiiPlusNT-LT Motion Controller

The SPiiPlusNT- LT motion controller is an economical 4 or 8-axis (local) controller network master. The controller can be housed in an MC4U or installed as a stand-alone. The controller consists of three interconnected boards: a base driver board, an MPU, and an Ethernet board. SPiiPlusNT- LT supports up to 32 axes and 64 nodes.

- > SPiiPlusNT- HP Motion Controller

The SPiiPlusNT- HP motion controller is a high performance, 8-axis (local) controller network master. The controller can be housed in an MC4U or installed as a stand-alone.

The controller consists of three interconnected boards: a base driver board, an MPU, and an Ethernet board. SPiiPlusNT- HP supports up to 64 axes and 64 nodes.

- > SPiiPlusNT-NP Motion Controller
The SPiiPlusNT-NP motion controller is a NanoPWM dual axis 2 or 4 (local) network master. The controller can be housed in an MC4U or installed as a stand-alone. The controller consists of three interconnected boards: a base driver board, an MPU, and an Ethernet board. SPiiPlusNT- NP supports up to 64 axes and 64 nodes.
- > SPiiPlusNT-LD Motion Controller
SPiiPlusNT-LD is the same as SPiiPlusNT-HP but includes an additional module for linear drivers.
- > SPiiPlusSC motion controller
SPiiPlusSC is a technology that leverages modern PC power and open architecture for high-performance motion control. ACS and non-ACS party servo drives and I/O devices are connected to the PC by means of real-time network EtherCAT. The SPiiPlusSC implements both multi-axes motion control and EtherCAT Master.

4.1.2 Supported Network Slaves

The ACS units that may be employed as slaves in the network are:

- > SPiiPlusDC motion controllers

The SPiiPlusDC product line consists of:

- > SPiiPlusDC- LT Motion Controller
The SPiiPlusDC- LT motion controller is an economical, 4 or 8-axis drive slave controller that can be incorporated in any network being controlled by any ACS EtherCAT master controller to which it is slaved. SPiiPlusDC- HP/LT motion controllers provide servo control over 4 or 8 local axis, and receives trajectories via EtherCAT protocol from the master SPiiPlusNT-HP.
- > SPiiPlusDC- HP
The SPiiPlusDC- HP motion controller is a high performance, 4 or 8-axis drive slave controller that can be incorporated in any network being controlled by any ACS EtherCAT master controller to which it is slaved. SPiiPlusDC- HP motion controllers provide servo control over 4/8 local axis, and receives trajectories via EtherCAT protocol from the master SPiiPlus NT-HP.
- > SPiiPlusDC-NP
The SPiiPlusDC- NP motion controller is a NanoPWM dual axis drive slave controller that can be incorporated in any network being controlled by any ACS EtherCAT master controller to which it is slaved. SPiiPlusDC- NP motion controllers provide servo control over 2/4 local axis, and receives trajectories via EtherCAT protocol from the master SPiiPlus NT-NP.
- > SPiiPlusDC-LD
SPiiPlusDC-LD is the same as SPiiPlusDC-HP but includes an additional module for linear drivers.
- > PDMnt Network Interface

The PDMnt is a network module designed for controlling external drives and I/Os. There are three PDMnt versions:

- > PDMnt-4 - A 4-axes Pulse/Dir interface for step and servo motors.
- > PDMnt-8/8-D - An 8 x digital output and 8 x digital input that can operate off 5Vdc or 24Vdc and can be configured either as a sink or source.
- > PDMnt-4-8/8-D - Combines the PDMnt-4 and PDMnt-8/8-D for providing a 4 x Pulse/Dir interface, 8 x digital inputs and 8 x outputs
- > SPiiPlus IOMnt EtherCAT Module

The SPiiPlus IOMnt EtherCAT Module is a 32 input and 32 output ACS EtherCAT network element. The product is configured, controlled and monitored by an ACS master and matching software tools.

- > SPiiPlus SDMnt Control Module

The SPiiPlus SDMnt Multi-Axis Step Motor EtherCAT Control Module is a panel mounted, four or eight axis EtherCAT slave module designed for running step motors. It can run seven two-phase unipolar step motors and one two-phase bipolar motor.

- > SPiiPlus UDMnt

The SPiiPlus UDMnt (Universal Drive Module) is a dual-axis card designed for incorporation in the MC4U Control Module to enable control peripherals over the Ethernet.

Details of the above units can be found in their respective Hardware Guide.

4.1.3 Supported Non-ACS Units

Currently the following non-ACS units have been certified for use within an ACS network:

- > Copley Controls Accelnet (Plus Panel Ethercat)
- > Sanyo Denki SanMotion RS2 with EtherCAT Interface
- > Yaskawa SGD5 Sigma 5
- > Emerson (Control Technix) Digitax ST

Non-ACS devices are divided into two main groups:

1. Qualified Motion Drives with high-level support:
 - > Servo
 - > Stepper
2. All other non-ACS slaves are classified as I/O devices.

4.1.4 EtherCAT Cable Length

The length of the cable between two adjacent network elements can be up to 100m using ACS certified components and cables. ACS has performed formal tests with 50m cables between adjacent network elements.

4.2 Numbering and Identification of Network Units, Axes and IOs

4.2.1 Unit ID Numbers

A network unit's identification (ID) is determined automatically by the system at power up following a master reset according to the physical order of the units in the network and in a sequential order.



A 'unit' is a network entity. As an example, an 8 axes MC4U is represented in the System Setup component as 'Network unit 0', or ID=0. Numbering starts at 0, up to the number of units in the network.

The IDs are assigned a number starting from 0. In some of ACS products, a physical switch exists on the product's panel: the DIP switch, which can be set by the user according to the following:

- > When set to 00 (binary) – this is a don't care setting.
- > When set to a non-zero value – it is checked by the system to detect changes of same-type units. This feature is used to distinguish between any identical and adjacently position units. In the System Setup MMI component the DIP is represented as DIP: 0 or other value.

The network assigns ID numbers to units sequentially (starting from 0) as a function of number of nodes.

4.2.2 Axes Numbering in ACSPL+

The firmware reads the units' axes and assigns them automatically, sequentially, according to the units' positions in the network. The units' configurations from the controller's and drives' memory, as stored in the master, are checked by the system at power up. A warning is issued in the following cases:

- > A unit has not been configured and thus does not have a configuration file
- > A conflict between the configuration file and the actual hardware settings read by the controller, hardware does not support the assignment.

Axes' assignment must be consecutive within each unit but may have gaps between the units and does not have to be ordered. The following assignments are valid:

MC4Unt-8	UDMnt-2	SDMnt-8
8-15	0, 1	19-22

When changing the order of units in the network, the axes' numbering (logical names that have been assigned) can be maintained. This is done by the System Setup MMI component, which checks the new assignments validity and warns of conflicts.

4.2.3 I/O Numbering in ACSPL+

- > Digital In and Out

Digital I/O variables **IN**(n) and **OUT**(n) are assigned sequentially by the System Setup MMI component as function of the detected hardware's number of IN and OUT ports.

As an example, a 32 bit vector assigned to a unit (32 in and out, maximum, in IOMnt, and as function of supported I/Os in other units), and addressed by bit: **IN**(3).5 where 3, the index, is set in the System Setup MMI component, and 5 is the bit number to which the default is assigned by the tool and can be changed by user. The variable **IN**(3).xx is assigned to a certain unit and cannot be split – no two units can have the same **IN**(X) assigned to them.

> Analog In and Out

Analog I/O variables **AIN**(n) and **AOUT**(n), n=0 to 999, are assigned sequentially by the System Setup component as function of the detected hardware's number of **AIN** and **AOUT** ports.

4.2.4 Logical and Physical Assignment of Network Units, Axes and IO Variables

Variables' names have to change when network units change locations. Since the units' position has changed, their IDs change, and, as a result, the axes assignment changes. In order to maintain the logical names of the axes and I/Os and thus leave the application unchanged, the System Setup MMI component data has to be updated.

As an example, the following 'before' and 'after' (UDMnt-1 has been removed from the network) configurations demonstrate the automatic axis assignment which results following the units' order change, and the assignment after user's intervention. An NTM master (not shown) is assumed:

Before:

	UDMnt-2	UDMnt-1	MC4U
ID	0	1	2
Axes	0, 1	2	3, 4, 5, 6

After:

	UDMnt-2	MC4U	Remarks
ID	0	1	
Axes	0, 1	2, 3, 4, 5	Default assignment
Axes	0, 1	3, 4, 5, 6	Assignment through System Setup MMI component by user

4.3 ACSPL+ Programming in a Network Environment

Some non-ACS network elements the commands: **ECIN** and **ECOUT**. These are used for low level programming which involve direct mapping of the EtherCAT telegram to user defined ACSPL+ variables.

Until version 2.60, controlling IOMNT units required using **ECIN** and **ECOUT**. ACSPL+ **IN** and **OUT** variables were not allocated to IOMNT. Version 2.60 supports auto-mapping of IOMNT to ACSPL+ **IN** and **OUT** variables.

For backwards compatibility, a new bit **#IOMNTMAP** in **S_SETUP** variable should be set to 1.

Refer to the *SPIiPlus Command & Variable Reference Guide*.

4.4 Network Operation

4.4.1 Power-Up Sequence

An ACS master has to be powered up, or reset, after all its slave units have reached their stable operational mode. This mode is reflected in the ACS master unit as the **MPU_ON** LED blinking green.

The network unit is identified and monitored in the following way:

- > An ACS master unit is monitored constantly, thus powering up or re-connecting a unit is detected immediately.
- > A non-ACS unit is detected only at the master's power up or reset process. As a result of reconnection or powering up a non-ACS unit it must be accompanied by restarting the network.

If the master is powered up after all the slaves are up, the master resets and reloads the network.

4.4.2 Cable Disconnection

After a cable is reconnected, all disconnected units are redetected and the **FCLEAR ALL** command has to be issued (see *SPIiPlus Command & Variable Reference Guide*). Encoder data of the disconnected units is maintained (as long as the unit has maintained its power), so that upon reconnection the drive's position is maintained.



It is the responsibility of the operator to ensure that the required precautions are taken prior to reconnecting the cables so that no personnel are endangered.

4.4.3 Losing 24V Power

If any unit has lost its 24V control power-feed, after being repowered, the entire network has to be restarted.

4.4.3.1 Network Monitoring

In order to perform network monitoring in order to check that all units are communicating with master, and verify the status of critical parameters, the following commands are available:

Command	Meaning
?ECST	Read EtherCAT State - a variable whose bits represent the various stages of the network
?ECERR	EtherCAT error - network error code
#LOG	Displays last 100 errors reported by the firmware

4.5 System Setup

ACS Motion Control's NT technology allows several units to be chained together using EtherCAT. **System Setup** enables you to configure a single or multiple Unit Control Module along with stand-alone NT modules and non-ACS units as a network.

Use **System Setup** to:

- > Define the system configuration
- > Update the system configuration

You can setup the system either automatically or manually.



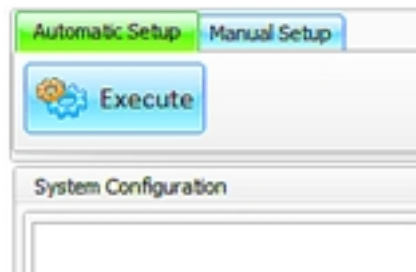
You should run the automatic system setup on initialization and whenever the system is changed.

4.5.1 Configuring the Network

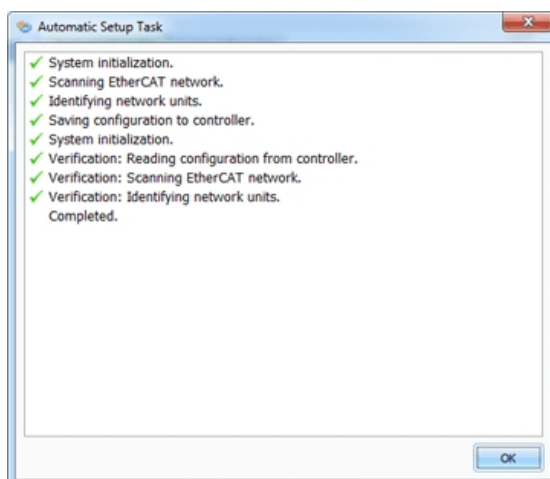
4.5.1.1 Automatic System Setup

To activate **System Setup**:

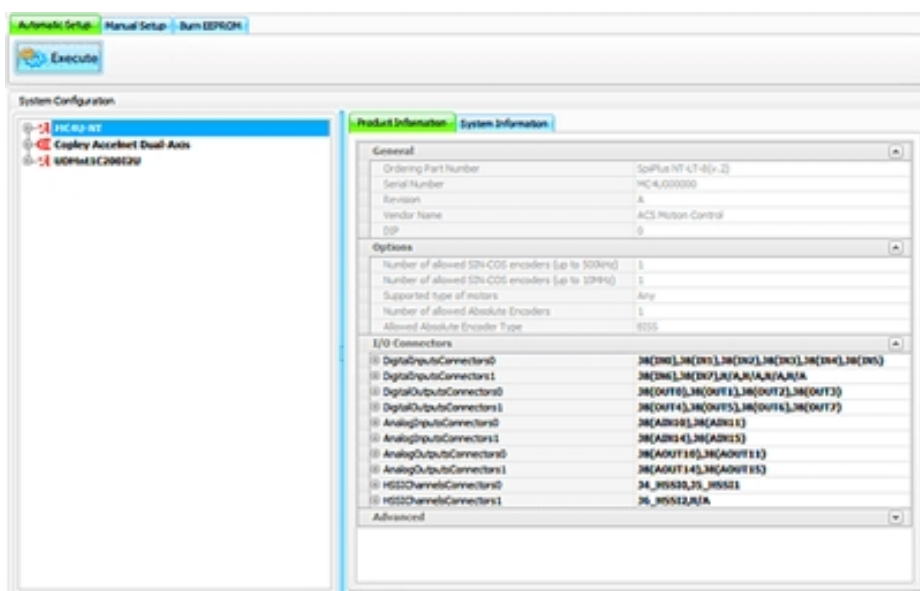
1. In the Workspace right-click the controller.
2. Select **Add component > Setup > System Setup**.
The System Configuration window opens.
3. In the Automatic Setup tab, press **Execute**.



4. Approve the message about the controller being rebooted.
The system is setup automatically. This may take several minutes to complete.
5. When the setup is complete, click **OK**.



The Setup results will look like this:



5. System Hardware Configuration

The SPiiPlus MMI Application Studio **System Configuration Wizard** is used for configuring the devices in a network. Your system hardware can be a single device (such as a single MC4U), or it may be a series of devices that are linked together through an EtherCAT network. There are three possible types of devices:

- > MC4U control modules
- > ACS Motion Control I/O modules (such as the SPiiPlus IOMnt) and Drive modules (such as the SPiiPlus UDMnt)
- > ACS-certified non-ACS modules (such as Sanyo Denki SanMotion RS2)

When configuring a new system, the wizard requires two steps:

- > Defining the hardware configuration in the database
- > Loading the hardware configuration into the master controller flash memory

6. Axis Setup and Tuning

Axis configuration and setup is accomplished using the SPiiPlus MMI Application Studio **Adjuster Wizard**. You use the **Adjuster Wizard** to set the parameters for the:

- > Axis architecture
- > Drive
- > Motor
- > Feedback

This chapter covers using the **Adjuster Wizard** to setup initial motion-related values for the SPiiPlus Motion Controller in the system. For complete details of using the **Adjuster Wizard** see the *SPiiPlus MMI Application Studio User Guide*.

6.1 Principles of Motion Control

A general motion control system can be divided into:

- > Controller – control laws
- > Drive – power converter
- > Machine – plant, motors and feedback devices

The plant receives two types of signals:

- > Controller output from the drive
- > Disturbances

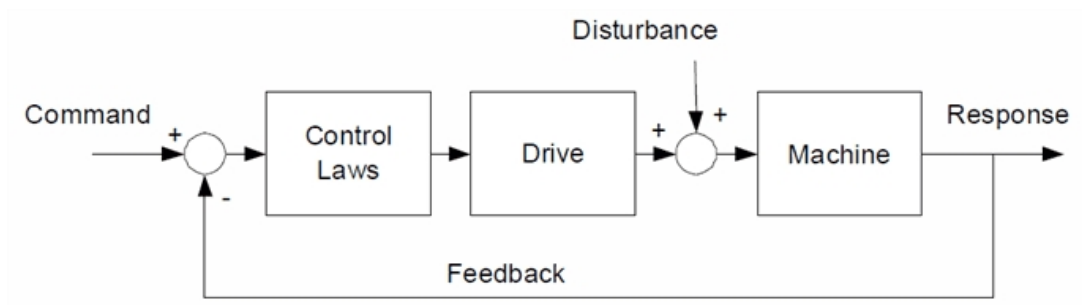


Figure 5-1. Motion Control System Schematic

Your goal in using the **Adjuster Wizard** is to make the plant follow the command “as good as possible” despite the presence of disturbances while ensuring that the system is stable. Through the **Adjuster Wizard** you adjust the parameters of the control laws in order to attain a quick, stable command response.

Control laws must be set with enough margins to accommodate reasonable changes in the system, and from system to another, such as a change in motor constant, driver gain, etc.

6.1.1 Control Loops

Ideal motion control is attained through properly tuned cascaded control loops:

- > Current loop (external / internal drive)

- > Velocity loop
- > Position loop

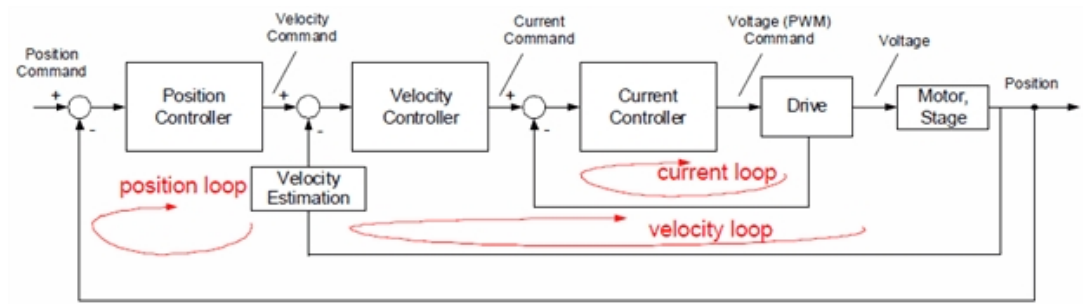


Figure 5-2. Control Loops

The position loop generates a command to the velocity loop, and the velocity loop generates a command to the current loop.

In general, an inner loop should be faster than an outer loop, specifically:

- > Current response should be faster than velocity response
- > Velocity response should be faster than position response

By faster we mean with higher bandwidths, at a higher frequency zone. Typical bandwidth values are:

- > Current loop – 0.5 kHz- 1 kHz
- > Velocity loop – 50 Hz – 200 Hz
- > Position loop – 10 Hz – 50 Hz

6.1.2 SPiiPlus Motion Control Algorithms

Elements used in the SPiiPlus control algorithm are:

- > PI filter
- > Low pass filter
- > Notch filter

6.1.2.1 PI Filter

The proportional-integral (**PI**) filter is a basic control element. It is included in the current loop and velocity loop. In certain cases also in the position loop.

The PI is the sum of two signals:

- > The **proportional** term (**SLxKP**) provides responsiveness, and affects the bandwidth (how fast the response is)
- > The **integral** term (**SLxKI**) ensures that the average error is driven to zero.

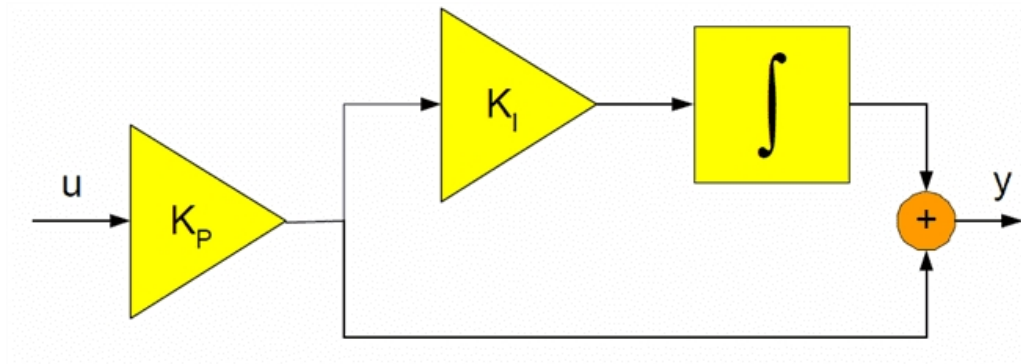


Figure 5-3. PI Filter Schematic

Figure 5-4 shows the frequency response of a PI filter.

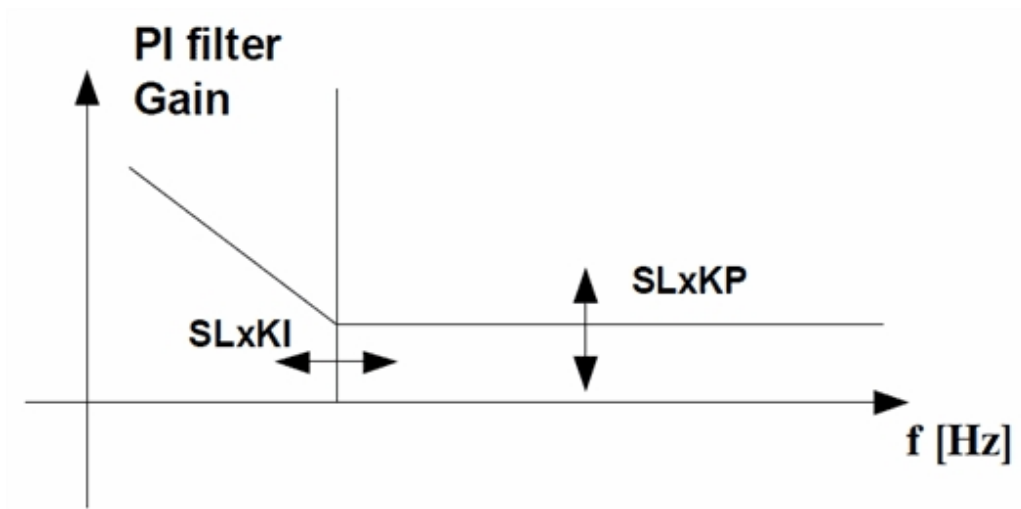


Figure 5-4. PI Filter Frequency Response

- > The proportional gain dominates the higher frequency zone.
- > The integrator gain dominates the lower frequency zone.
- > The frequency of the PI zero is always at: $SLxKI/20$ Hz

6.1.2.2 Low Pass Filter

SPiiPlus provides a second order low-pass filter. In many cases, it can be used to increase the overall bandwidth of the velocity loop. It attenuates high frequency noise and resonances (a phenomenon in which the plant has high gain around one frequency).

The second order low-pass filter attenuates all signal components above a certain frequency. The frequency is specified by the **SLVSOF** variable.

Filter damping is determined by the **SLVSOFD** variable.

In many cases a proper reduction of the low pass filter bandwidth (relative to its default value) allows to further increase the velocity gain. The major disadvantage of the second order low pass filter is that it affects stability by adding a significant amount of phase lag and thus reducing the phase margin.

Usually, **SLVSOF** has to be 10-20 times above the velocity loop bandwidth. As a rule of thumb: set **SLVSOF=SLVKI**.

6.1.2.3 Notch Filter

SPiiPlus provides a Notch Filter. In many cases, it can be used to increase the overall bandwidth of the velocity loop. The Notch filter attenuates only a narrow band of frequencies and is usually set above the bandwidth of the velocity loop.

Notch Filter advantage is less phase lag contribution in comparison to the low-pass filter. The Notch Filter disadvantages are:

- > Usually the transfer function of the plant has to be known in order to place the Notch Filter properly.
- > The Notch bandwidth has to be sufficiently wide. Resonant frequencies may vary slightly in different machines of the same kind. The resonant frequency may also depend on the position of the axes.

The Notch filter parameters are:

- > **SLVNFRQ** – Notch frequency
- > **SLVNWID** – Notch width (3 dB points)
- > **SLVNATT** – Attenuation (absolute units)

6.1.2.4 Bi-Quad Filter

A Bi-Quad filter is added to the velocity loop control in addition to the existing 2nd order Low-pass and Notch filters.

The Bi-Quad filter is the most general 2nd order filter. It has two poles and two zeros. It can be thought of as a high-pass filter in series with a low-pass filter. The transfer function of the Bi-Quad filter is as follows:

$$H(s) = \frac{(s/\omega_N)^2 + 2\zeta_N(s/\omega_N) + 1}{(s/\omega_D)^2 + 2\zeta_D(s/\omega_D) + 1}$$

Where:

- > ω_N and ω_D are the numerator (high-pass filter zero) and denominator (low-pass filter pole) frequencies, respectively.
- > ζ_N and ζ_D are the numerator and denominator damping ratios, respectively.

The corresponding ACSPL+ parameters are:

- > **SLVBONF, SLVBODF** - numerator and denominator frequencies in [Hz]. Range: 0.1 - 4000 [Hz].
- > **SLVBOND, SLVBODD** - numerator and denominator damping ratios. Range: 0.01 - 1.
- > **MFLAGS** bit 16 is used to activate the filter (by default the filter is off).

The Bi-Quad filter can be used to compensate mechanical resonances, improve stability margins and system bandwidth.

The Bi-Quad filter can be configured as an additional Notch using the following formulas:

1. Set the numerator and denominator frequencies equal to the Notch frequency in[Hz]:

$$SLVB0NF = SLVB0DF = SLVNFRQ[Hz]$$

2. Set the numerator damping ratio equal to half the ratio between the Notch width and Notch frequency:

$$SLVB0ND = \frac{SLVNWID[Hz]}{2 \times SLVNFRQ[Hz]}$$

Maximal recommended ratio between width and frequency= 1/3.

3. Set the denominator damping ratio equal the numerator damping ratio times the Notch attenuation (in absolute value):

$$SLVB0DD = SLVB0ND \times SLVNATT$$

Below there are several examples that demonstrate the generality of the Bi-Quad filter.

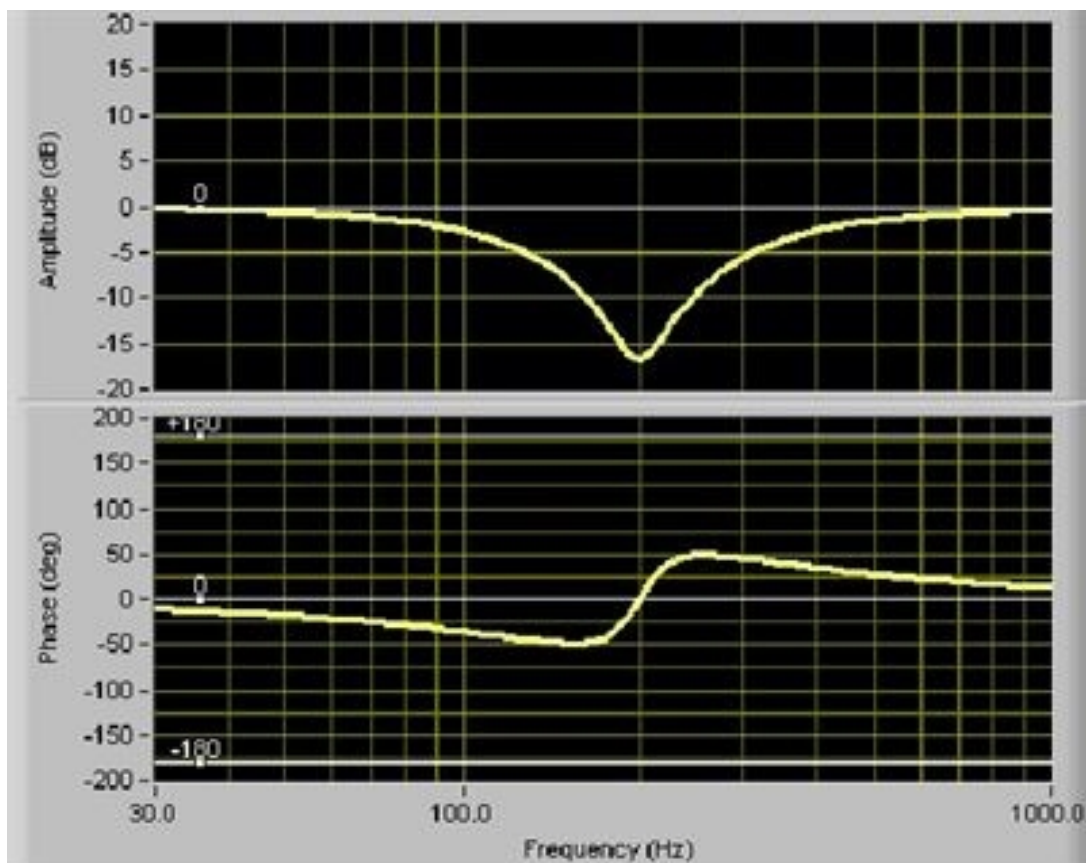


Figure 5-5. Bi-Quad Configured as a Notch Filter

$$SLVB0NF = SLVB0DF$$

$$SLVB0ND < SLVB0DD$$

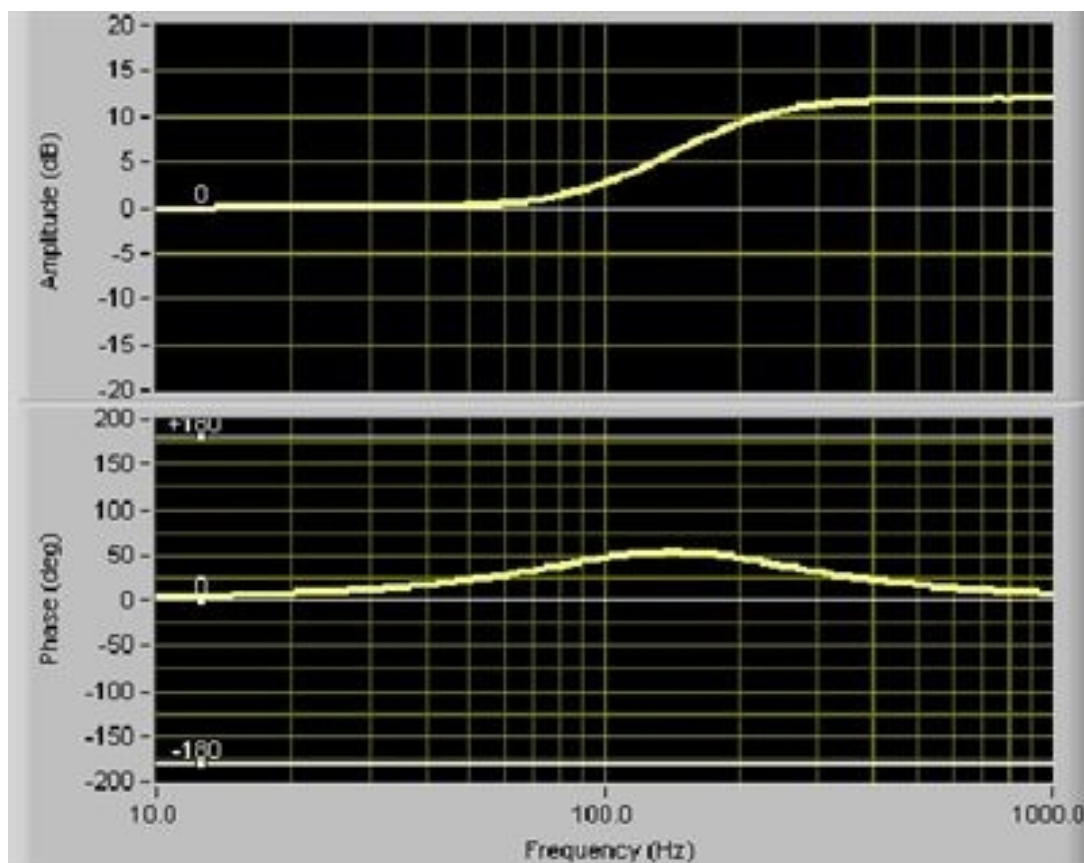


Figure 5-6. Bi-Quad Configured as a 2nd Order Lead Filter

This kind of filter is used to improve the phase margin

SLVBONF < SLVBODF

In the example the damping ratios are equal:

SLVBOND=SLVBODD =0.707

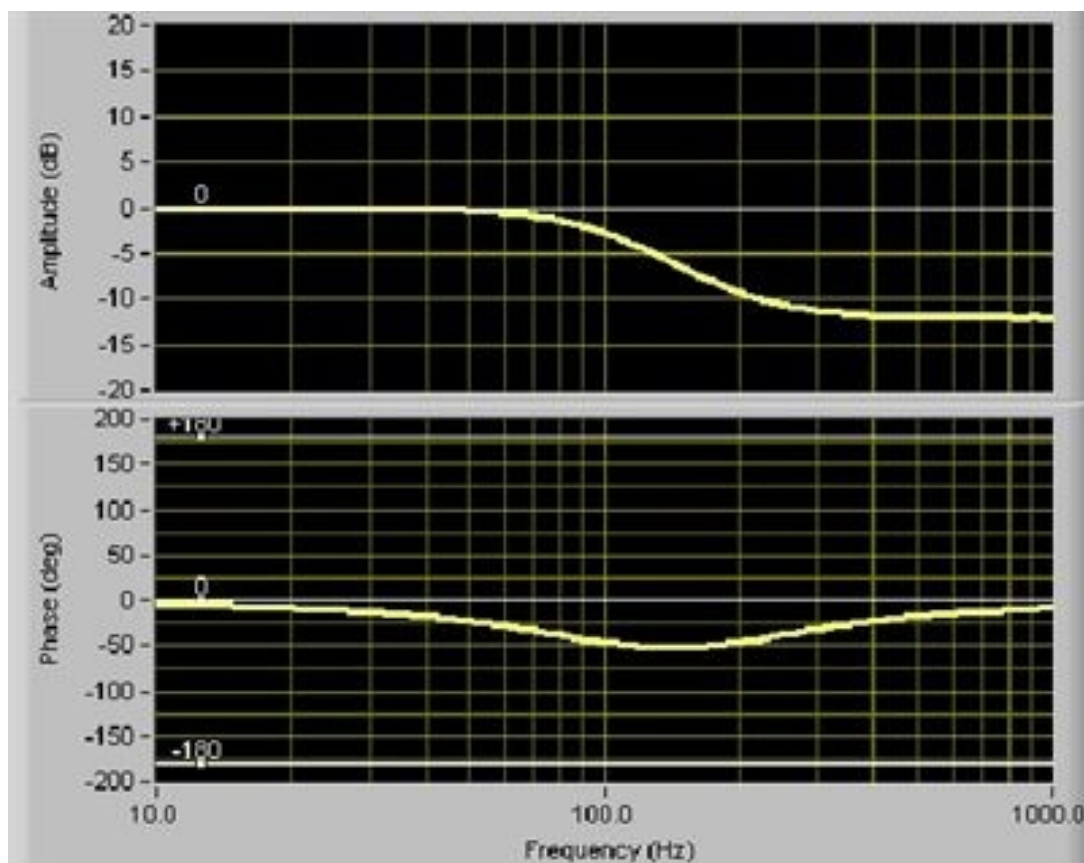


Figure 5-7. Bi-Quad Configured as a 2nd Order Lag Filter

This kind of filter is used to improve the gain margin

SLVBONF > SLVBODF

In the example the damping ratios are equal:

SLVBOND=SLVBODD =0.707

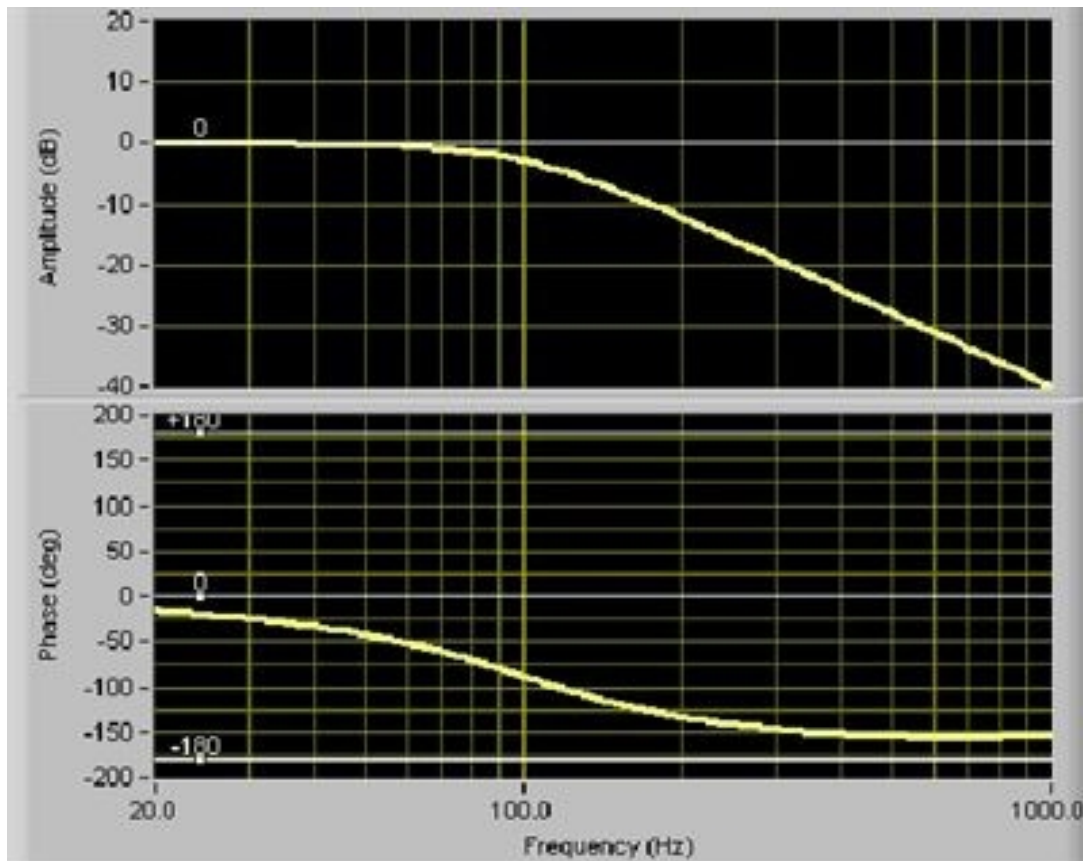


Figure 5-8. Bi-Quad Configured as a 2nd Order Low Pass Filter

SLVBONF > SLVBODF

6.2 About the Adjuster Wizard

The **Adjuster Wizard** provides tuning in terms of high-level parameters, namely parameters that have a clear meaning for you. Once you have assigned high-level parameters, the **Adjuster Wizard** automatically transforms them into low-level parameters, that is, controller parameters.

Since the **Adjuster Wizard** works together with the SPiiPlus MMI Application Studio **Scope** monitoring tool, it provides interactive tuning. The **Adjuster Wizard** sends a trial signal to the motor and automatically selects suitable **Scope** settings and sends the low-level parameter values. The **Scope** displays the motor response and you can immediately view the results.

The controller needs to be configured and adjusted for each axis. The adjustment procedure affects both the volatile and non-volatile memory of the controller. The **Adjuster Wizard** maintains the high-level adjusted values for each component separately on your computer. Only after you have selected Save to Flash, does the **Adjuster Wizard** save the values into low-level parameters in the controller.

Some operations during the adjustment process may cause the motor to begin moving unexpectedly. Some of the limits and safety features may be temporarily disabled. To avoid personal injury or damage to the equipment, check the following before starting the adjustment process:



- > Verify that NOTHING (people, electrical cables, or other obstacles) is in the path of the motor or objects connected to the motor.
- > Verify that the motor is securely anchored and that proper safety barriers, stops, and/or limits are installed.



You must run the **Adjuster Wizard** for each new SPiiPlus Motion Controller in the system.

The adjustment procedure affects both the volatile and non-volatile memory of the controller. The **Adjuster Wizard** keeps the controller memory in-sync with the application database.

The **Adjuster Wizard** maintains the high-level adjusted values for each component separately on your computer. Only after you have selected **Save to Flash**, does the **Adjuster Wizard** save the values into low-level parameters in the controller.







If the controller is shut off during an adjustment session and the data has not been saved to the controller's flash memory, the synchronization will be lost and the adjustment session must be repeated.

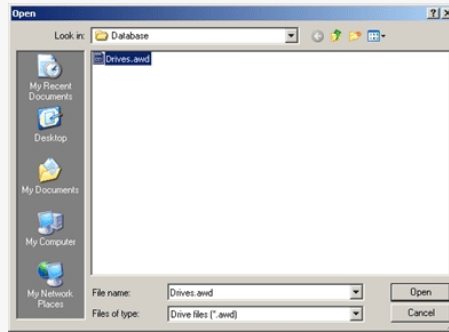
6.2.1 Working in Adjuster Wizard Task Windows

There are several elements that are common to most of the **Adjuster Wizard** task windows. These are:

6.2.1.1 Data Action Buttons

There are five data action buttons:

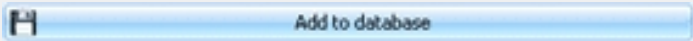


	Add an item to the field.
	Edit the data fields.
	Deletes the item from the database.
	Import data from a previously saved file on the computer. When you click this button, a browser window is displayed, for example:



Drill down to the subdirectory containing database files and select the file, all Adjuster database files have an **awd** file extension. Then click **Open**.

6.2.1.2 Adding Data to Component Database

When you define the values for the Motor, Drive and Feedback components, the following action buttons are available:

	<p>Active if you have entered a component name that is not in the database. Clicking this button adds the data to the Component database under the component name.</p>
	<p>Active if you have made changes to the data that exists in the database. Clicking this button puts changes into the Component database.</p>
	<p>Deletes all changes you have made to the data fields.</p>

6.2.1.3 Incompatibility Symbol

The **Incompatibility Symbol**  indicates that the value in the component field is incompatible with what you have defined in previous windows.

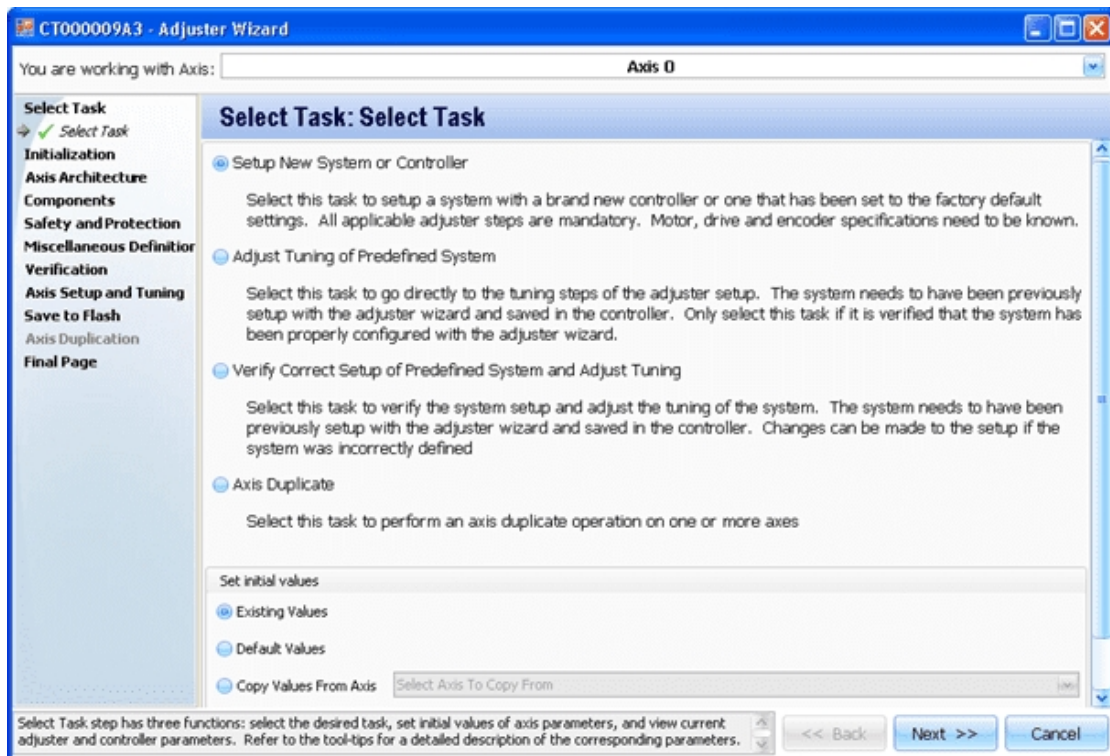
You should return to a previous window and change the value with which it is in conflict, or, if possible, change the value in the field in the current window. In any case you should resolve all conflicts before continuing.



This symbol can be ignored when an element is not yet defined. For example, a Drive has been defined but not the Motor.

6.3 Starting Adjuster Wizard

1. In the Toolbox click **Setup** to display the Setup list of tools.
2. Click **Adjuster Wizard** in the list of tools.
The Adjuster Wizard **Select Task** window is displayed in the workspace.



You can also activate the **Adjuster Wizard** using the right-click **Add Component** option of the Workspace Tree.

3. Select the axis by clicking the Axis Selector field and then select the axis from the dropdown list:



The Adjuster Wizard must be run for each axis in the system.

6.4 Selecting Axis Setup Initial Values

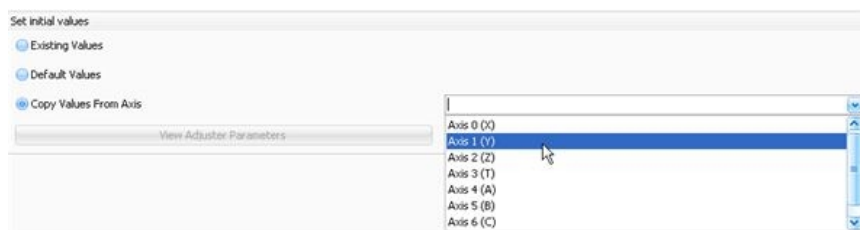
You use the **Set initial values** option to set the values that the **Adjuster** is to use at the start of the adjustment process: You have two choices:

- > **Existing Values** - using the values that have been built into the **Adjuster Wizard** database.
- > **Default Values** - using factory default values that have been set in the controller.

When you select this option, a prompt window is displayed:



If you have already run the **Adjuster** for another axis, you have the option of copying values from this axis into the current axis by selecting **Copy Values From Axis**. When you select this option, you choose the axis whose values are to be used from the **Select Axis To Copy From** (which becomes active when you select this option) dropdown list:



Click **Yes** to load the values.

To setup the axis values select **Setup New System or Controller** from the Task window and click **Next**. There are nine steps in this task:

Initialization

In this step you provide information that serves to identify the database containing the high-level parameter values.

Axis Architecture

In this step you define the electro-mechanical structure of the controller's working application as well as the units for measuring feedback.

Components

In this step you define the parameters governing the:

- > Drive
- > Motor
- > Feedback

Once you have defined these, you trigger the **Adjuster** to calculate the optimal values for the high-level parameters and you load them into the system.

Safety and Protection

In this step you define the parameters for:

- > Motion limits
- > Current limits
- > Position errors
- > Position limits

Miscellaneous Definitions

In this step you define the parameters for such things as:

- > Motion completion
- > Enable/Disable/Brake
- > Dynamic brake
- > Home switch

Verification

In this step the Adjuster Wizard runs various verification tests for such things as:

- > Feedback
- > Motor
- > Switches
- > Stop, Alarm, and Brake

This enables you to see where you need to adjust the values of the parameters.

Axis Setup and Tuning

In this step you are given the opportunity to refine your values and fine tune your system.

Save to Flash

Once your high-level values are correct for your system, in this step you save the low-level values to the controller flash.

Final Page

This step is your exit point from the **Adjuster Wizard**.

6.5 Initialization

Initialization serves only for tracking purposes.

1. Enter your name in the **User Name** field (optional - this field may be skipped).
2. Enter the name of your application or the ID of the machine for which the controller is being employed in the **Application/Machine** field (optional - this field may be skipped).



You will note that Adjuster has extracted the controller's firmware version, S/N and P/N from the controller and filled in these fields.

3. Enter the date in the **Date** field - its format is: mm/dd/yyyy.
4. Enter any remarks you want in the **Remarks** field (optional - this field may be skipped).
5. Click **Next**.

6.6 Axis Architecture

In this step you are to define the general parameters of the feedback mechanism to include the electro-mechanical characteristics of the motion and how feedback is to be measured and from what.

Data must be entered in all the fields. The data is entered by selecting the appropriate value from the dropdown list of each field.

#New Board - Adjuster Wizard

You are working with Axis: **Axis 0 (X)**

Select Task

- Initialization
- Axis Architecture**
 - Axis Structure
- Components
- Safety and Protection
- Miscellaneous Definition
- Verification
- Axis Setup and Tuning
- Save to flash
- Final Page

Axis Architecture: Axis Structure

Axis Structure

Axis Structure: Single Motor

Motor-Load Topology: Rotary Motor and Rotary Load - Direct Drive

Gear Ratio: 1

Feedback Topology: Single, on motor

User Units

User Units Applied To: Motor

Rotary Units

Rotary Units: Count - Encoder Count

☒ Advanced Parameters

Axis Structure step has two functions: define electro-mechanical structure of application and define user units used in motion programming. Refer to the tool-tips for a detailed description of the corresponding parameters.

<< Back Next >> Cancel

Once you have entered all the required data for this step, click **Next** to advance to the next task.

6.6.1 User Units

User Units

Of particular import in this step is the setting of User Units to be used by **Adjuster Wizard** in measuring feedback. 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 $2\text{mm}/4 = 0.5\text{mm}$.

The User Unit is applied to the **EFAC** variable. **EFAC** is a 64 member (one for each possible axis) real array used for defining a factor between the raw feedback in encoder counts and the **FPOS** value calculated by the controller. See *SPiiPlus Command & Variable Reference Guide* for complete details.

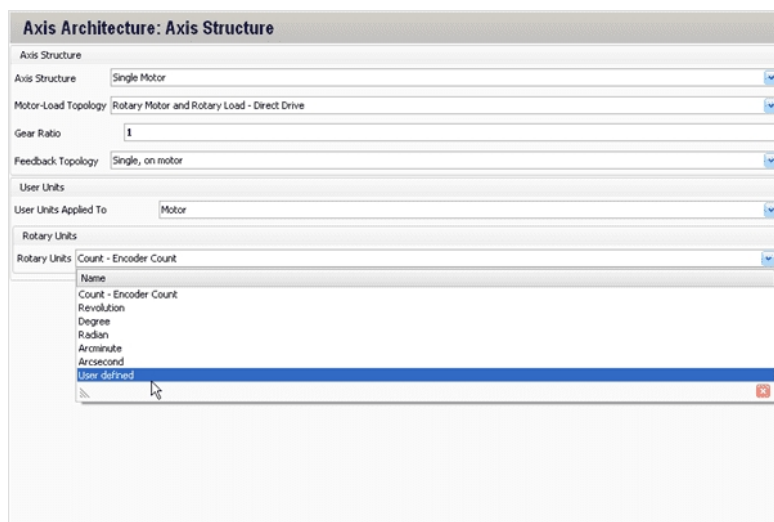
Examples of where User Units can be useful:

- > 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.

6.6.2 User Defined User Unit

You use **Advanced Parameters** to make your own definitions of the User Unit.

To display **Advanced Parameters** select **User defined** from the Units dropdown list, for example:



The **Advanced Parameters** panel is displayed:



The Advanced Parameters option should be used with caution. Badly defined units may cause serious problems with measuring the actual feedback.

1. Enter a unique parameter name in **User Defined Unit Name**.
2. Enter a mathematical expression in **Function**, you can use the standard arithmetical operations:
 - > + (Add)
 - > - (Subtract)
 - > * (Multiply)
 - > / (Divide)

The function is used as a conversion factor applied to the selected standard unit. For example: $2.71828 * 1.41421$ (e times the square root of 2).

3. Select a standard unit from the dropdown list in the **of** field.
4. Enter a correction factor (if different than 1) in **Correction Factor**. The correction factor is used to correct the difference between actual movement (as measured by a laser interferometer) and the software commanded move (indicated in the user units).



The EFAC variable relates to many other configuration variables. While changing the EFAC variable via the Adjuster, all relevant configuration variables are automatically updated. Therefore, we recommend changing the EFAC variable only via the Adjuster and not from other places, such as SPiiPlus MMI Application Studio Communication Terminal or any application program

6.7 Components

The Components step consists of defining the values for:

- > Motor
- > Drive
- > Feedback

At the end of the step the **Adjuster Wizard** calculates the parameters that are to be used and inserts them into the appropriate program variables.



The screen shots presented in this step are for example only. The actual fields that appear depend on the values set in the Axis Architecture step. See *SPiiPlus MMI Application Studio User Guide* for details of working with these windows.

6.7.1 Motor

The first component for which you need to enter values is the Motor.

The screenshot shows the 'Components: Motor' window. On the left is a sidebar with a tree view containing: Select Task, Initialization, Axis Architecture, Components (with Motor, Drive, Feedback, Motor, Calculate Parameters), Safety and Protection, Miscellaneous Definition, Verification, Axis Setup and Tuning, Save to flash, and Final Page. The main area is titled 'Components: Motor' and lists the following parameters for 'Axis 0 (X)':

- Motor:
- Topology:
- Type:
- Peak Current [A]*:
- Nominal Current [A]*:
- Maximum Velocity*: (Unit: RPM)
- Number of Poles*:
- Back EMF Constant: (Unit: V/(RPM))
- Phase Connection:
- Phase Resistance [Ω]:
- Serial Number:
- Remarks:

At the bottom, there are buttons: 'Update in database', 'Discard changes', '<< Back', 'Next >>', and 'Cancel'. A small note at the bottom left states: 'Components: Motor step provides the interface to define the motor parameters for the motor being used in the current Adjuster session. If certain motor parameters are not known refer to the tool-tips for guidance on how to approximate the values. It is highly recommended to specify the motor'.

Data must be entered in all the fields. The data is entered by selecting the appropriate value from the dropdown list of each field or entering the values taken from the motor's technical data sheet.



In entering the values the incompatibility icon (🚫) may appear. Check the tool-tip. If it states that there is incompatibility between components that have not been set yet, ignore it. Continue to set component values. However, do not forget to go back after all the components have been set up and ensure that it no longer appears. If it still appears, then resolve it by changing the component's values or changing the **Axis Setup** values.

Once you have entered all the values, click **Next** to define the Drive component.

6.7.2 Drive

The second component for which you need to enter values is the Drive.

The screenshot shows the 'Components: Drive' configuration window. The left sidebar lists the following tasks: Select Task, Initialization, Axis Architecture, Components (selected), Feedback, Motor, Calculate Parameters, Safety and Protection, Miscellaneous Definition, Verification, Axis Setup and Tuning, Save to flash, and Final Page. The 'Components' section is expanded, showing a list of components: Motor, Drive (selected), Feedback, Motor, and Calculate Parameters. The main area displays the 'Components: Drive' configuration for 'Axis 0 (X)'. The parameters are as follows:

Parameter	Value
Drive	Balloon
Controller-Drive Interface	Analog (±10V)
Type	Three phase, commutation by drive
Peak Current [A]*	10
Nominal Current [A]*	5
Supply Input Voltage Type	AC Voltage
Supply Input Voltage [V]*	24
Frequency [Hz]*	60
Serial Number	SB-55667
Remarks	

At the bottom of the window, there are buttons for 'Update in database', 'Discard changes', '<< Back', 'Next >>', and 'Cancel'. A small note at the bottom states: 'Components: Drive step provides the interface to define the drive parameters for the drive being used in the current Adjuster session. If certain drive parameters are not known refer to the tool-tips for guidance on how to approximate the values. Be sure that the selected drive is compatible.'

Data must be entered in all the fields. The data is entered by selecting the appropriate value from the dropdown list of each field or entering the values taken from the drive's technical data sheet.



In entering the values the incompatibility icon (🚫) may appear. Check the tool-tip. If it states that there is incompatibility between components that have not been set yet, ignore it. Continue to set component values. However, do not forget to go back after all the components have been set up and ensure that it no longer appears. If it still appears, then resolve it by changing the component's values or changing the **Axis Setup** values.

Once you have entered all the values, click **Next** to define the Feedback component.

6.7.3 Feedback

The third component for which you need to enter values is the feedback.



The **Feedback** component can be either the Motor, the Load, or both depending on what you defined in **Feedback Topology** in the [Axis Architecture](#) step.

Data must be entered in all the fields that are active (many, you will note, have already been selected by Adjuster from data that you have previously entered for the Drive and Motor components). The data is entered by selecting the appropriate value from the dropdown list of each field or entering the values directly in the fields.

For absolute encoders, see [Absolute Encoder Setup Procedure](#).



In entering the values the incompatibility icon (🚫) may appear. Check the tool-tip to see what is causing the incompatibility. If it is a value of a previously set up component, then return to the component and either change the component's value, or make changes in the **Axis Setup**.



Make sure that all component incompatibilities are resolved before continuing.

Click **Next** to advance to calculate and apply the feedback parameters.

6.7.3.1 Absolute Encoder Setup Procedure

To setup the absolute encoder, use the following procedure:

1. Prior to the setup, verify that the controller has a license for the specific encoder type, as well as hardware support. This can be verified by using the SPiiPlus MMI Application Studio **System Information Viewer** or the **#SI** command.
2. Use the SPiiPlus MMI Application Studio **Adjuster Wizard** to do the following:
 - > Up to the **Components:Feedback** step, setup the axis normally.
 - > In the **Components:Feedback** step, type the encoder parameters.
 - > Set the encoder parameters as follows:



For details of the variables and their permissible values, see the SPiiPlus ACSPL+ Command & Variable Reference Guide.

- > **Type Absolute Encoder BiSS-C:** in the **Communication Terminal**, set the absolute encoder BiSS-specific parameters **E_PAR_A** and **E_PAR_B** according to technical documentation for the encoder.
 - > For **Type Absolute Encoder HIPERFACE:** in the **Communication Terminal**, set the absolute encoder HIPERFACE-specific parameter **E_SCMUL** according to technical documentation for the encoder.
 - > Return to the **Adjuster Wizard** and click **Next**.
 - > In the **Calculate Parameters** step, click **Calculate Parameters** and then click **Apply changes**.
3. Continue to setup the axis using the Adjuster Wizard.

6.7.4 Calculate Parameters

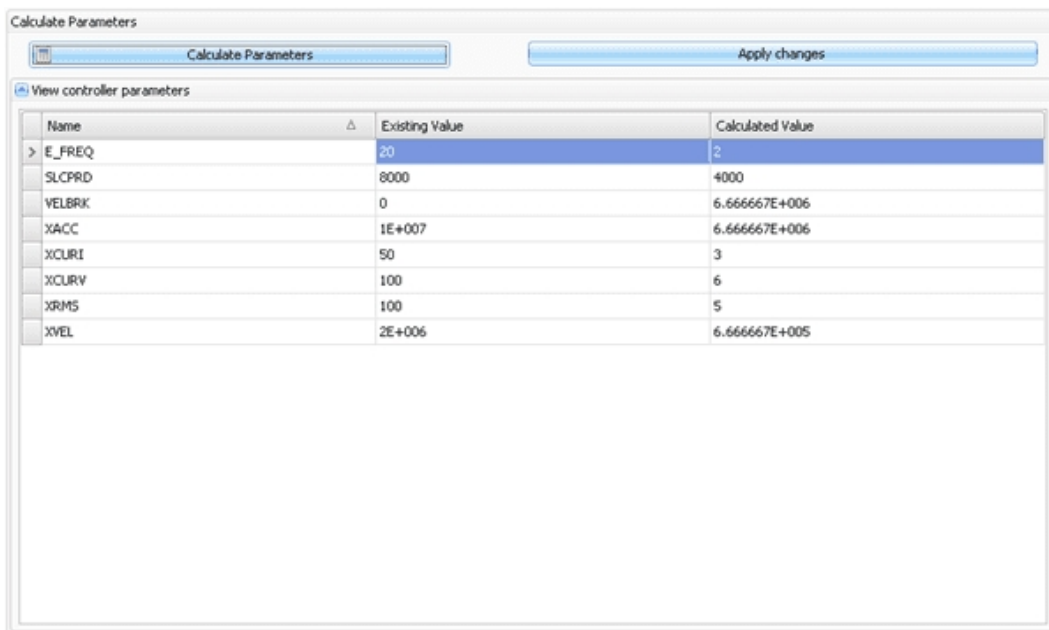
At this stage, the **Adjuster Wizard** verifies that the Axis Architecture, Motor, Drive and Feedback definitions are all compatible with each other.

If an inconsistency is detected, a prompt is displayed about it with recommended solutions.

If there are no inconsistencies, click:



The **Adjuster Wizard** calculates the values for the controller parameters and displays a table with each parameter name, its old and its new value, for example:



Click:  and click **Next** to advance to the next step.

6.8 Safety and Protection

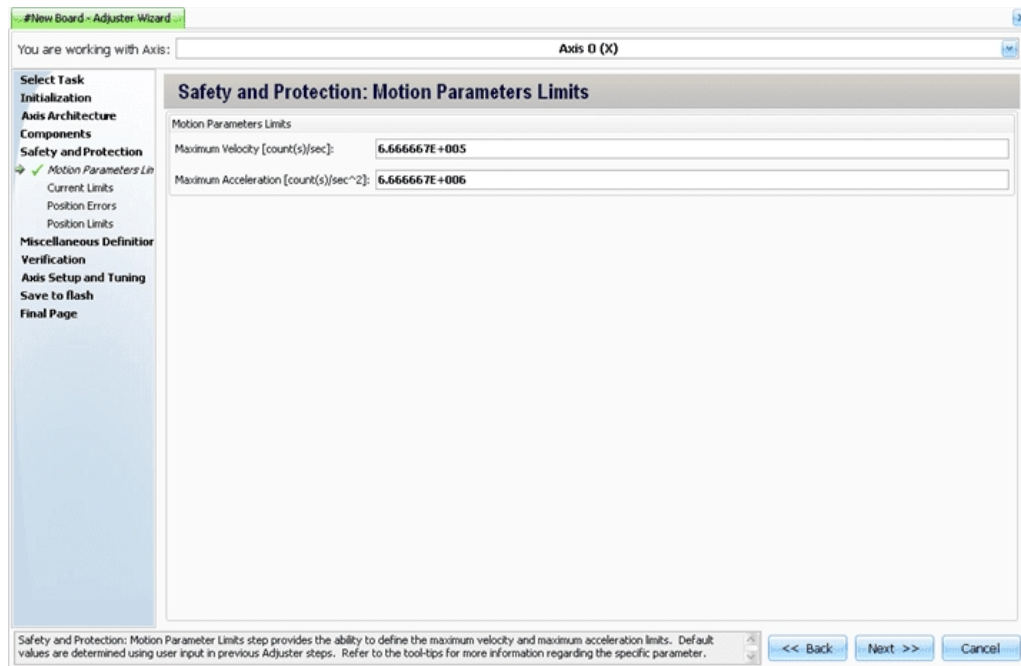
The Safety and Protection step enables you to set certain safety and protection variables. There are a total of four possible types of Safety and Protection parameters that you can set.



The actual number of parameter types and parameters available to you depends on what you defined in the **Axis Architecture** and **Components** steps.

The **Adjuster Wizard** displays values it has calculated in the window of each type. You have the option of changing these within the limits of valid values. For details of the variables and their permissible values see the *SPiiPlus ACSPL+ Command & Variable Reference Guide*.

6.8.1 Motion Parameter Limits



This window is used for entering the Maximum Velocity and Maximum Acceleration Limits values.

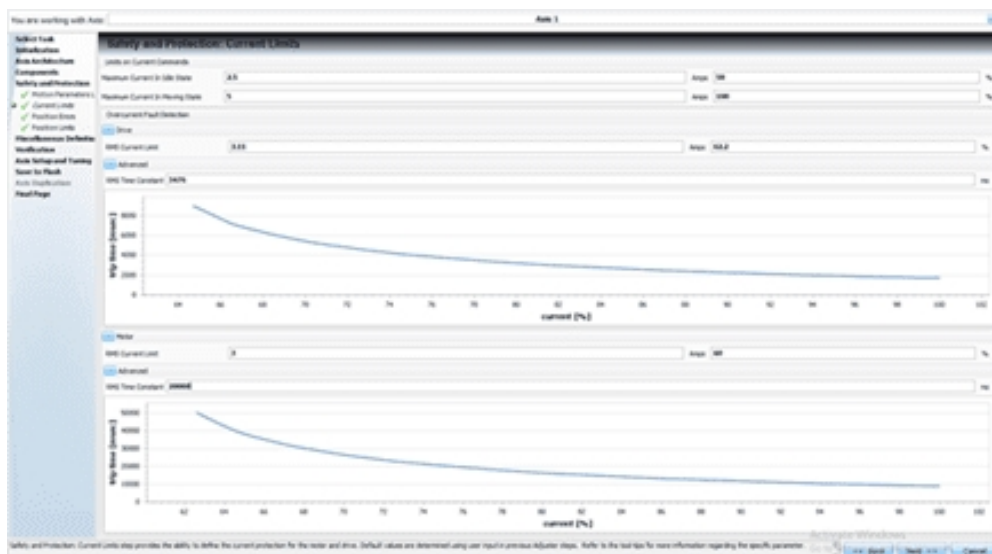
- > **MAXIMUM VELOCITY (XVEL) [UNITS/SEC]** - The maximum allowed value is automatically calculated based on motor parameters that were defined in [Components](#) step. Letting the mouse cursor linger on the field displays a tool-tip guiding you in permissible values.



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.

- > **MAXIMUM ACCELERATION (XACC) [UNITS/SEC²]** - The Adjuster puts in a value based on motor parameters that were defined in [Components](#) step. Letting the mouse cursor linger on the field displays a tool-tip guiding you in permissible values.

6.8.2 Current Limits



This window is used for defining motor and drive protection parameters:

- > **Maximum Current In Idle State (XCURI)** - The **Adjuster Wizard** automatically calculates the value from the maximum torque/force required during standstill to overcome gravitation, friction, etc.
If the value cannot be automatically calculated, that is, you have not specified gravitation, friction, etc., the parameter **XCURI** is set to default.
If you want to set values other than the value displayed, letting the mouse cursor linger on the field displays a tool-tip guiding you in permissible values. 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.
- > **Maximum Current In Moving State (XCURV)** - The value is automatically calculated according to minimum of either maximal acceleration (**XACC**) or motor peak current that you have specified.
If you want to set values other than the calculated one, letting the mouse cursor linger on the field displays a tool-tip providing guidance in changing maximal acceleration or motor peak values. $\text{XCURV} = [\text{allowed motor peak current} / \text{drive peak current}] \times 100$. Refer to the drive and motor specifications for these peak values.
- > **Overcurrent Fault detection** - The following numeric controls defines Overcurrent Fault detection criterion:
 - > **RMS CURRENT LIMIT (XRMS, XRMSD, XRMSM)** - The value is automatically calculated based on motor/drive parameters that were defined in the [Components](#) step.
If you want to set values other than the calculated ones, letting the mouse cursor linger on the field displays a tool-tip guiding you in permissible values for the motor and/or drive current limits. $\text{XRMS, XRMSD, or XRMSM} = [\text{allowed motor nominal current} / \text{drive peak current}] \times 100$. Refer to the drive and motor specifications for these values.

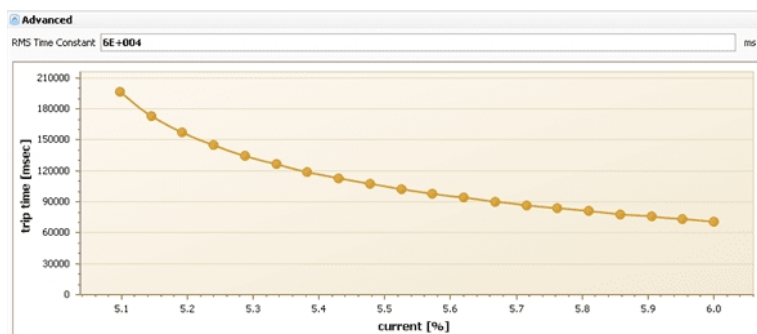
- > **RMS CURRENT TIME CONSTANT (XRMST, XRMSTD, XRMSTM) [MS]** - This is the time interval for checking whether an overcurrent fault has occurred. The overcurrent fault activation time is according to the following formula:

$$faultactivation = \ln \left((1 - X_{RMS}/D_{COM})^2 \right) \times X_{RMST}$$

For example: if **X_{RMS}**=50, **D_{COM}** = 100 and **X_{RMST}** = 3300 msec, then fault activation time = -ln (0.75)*3300 = 950msec.

To calculate the fault activation time for **X_{RMSD}** use **X_{RMSTD}**; for **X_{RMSM}** use **X_{RMSTM}**.

If you click the **Advanced** down arrow, the Adjuster displays a graph of the Overcurrent Fault Detection:



When you change parameter values, you can see the effect on the graph.

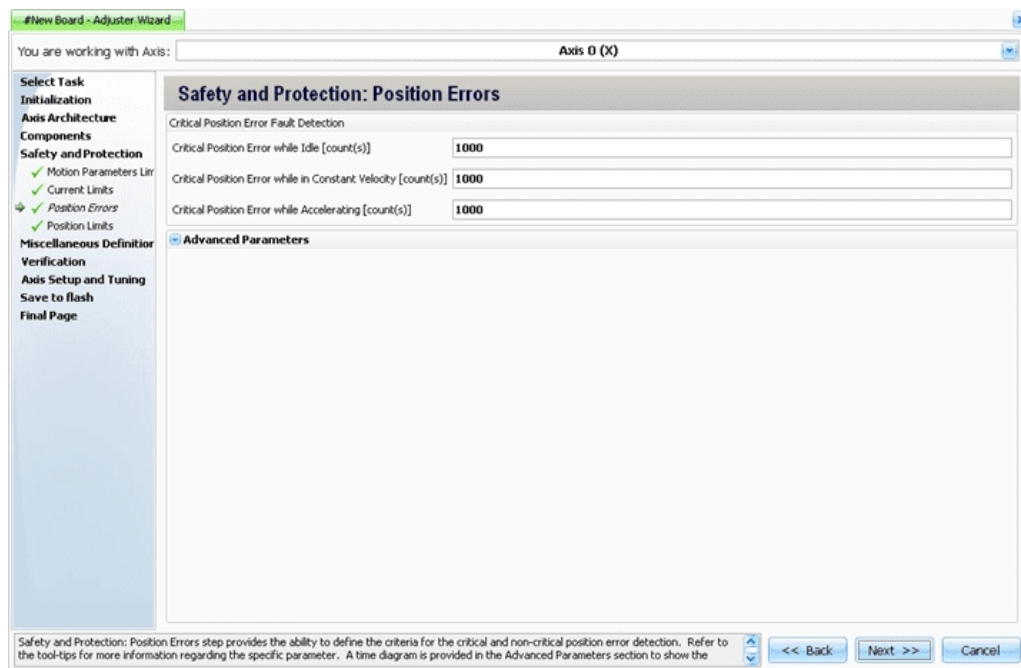
Setting current (torque) limits example:

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

In this case:

- > **X_{CURI}** = minimum current level required by the motor to keep the axis in position, and less than X_{CURV} (which is 50%).
- > **X_{CURV}** = (10/20)*100 = 50%.
- > **X_{RMS}** = (3/20)*100 = 15%.

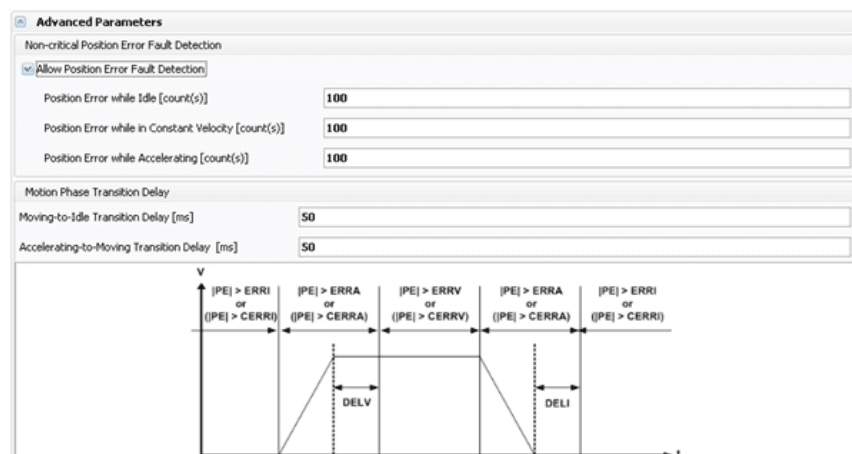
6.8.3 Position Errors



This window is used for defining Critical Position Errors criteria:

- > Critical Position Error While Idle (**CERRI**)
- > Critical Position Error While In Constant Velocity (**CERRV**)
- > Critical Position Error While Accelerating (**CERRA**)

If you click the **Advanced** down arrow, the Adjuster displays the following:



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.

- > **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 the idle state), **CERRI** (critical position error in the idle state).
- > Constant velocity: **ERRV** (position error in while in the constant motion state), **CERRV** (critical position error in while in the constant motion state)
- > Acceleration: **ERRA** (position error in while in the accelerating state), **CERRA** (critical position error in while in the accelerating state)

For example, when the motor is in the 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**.

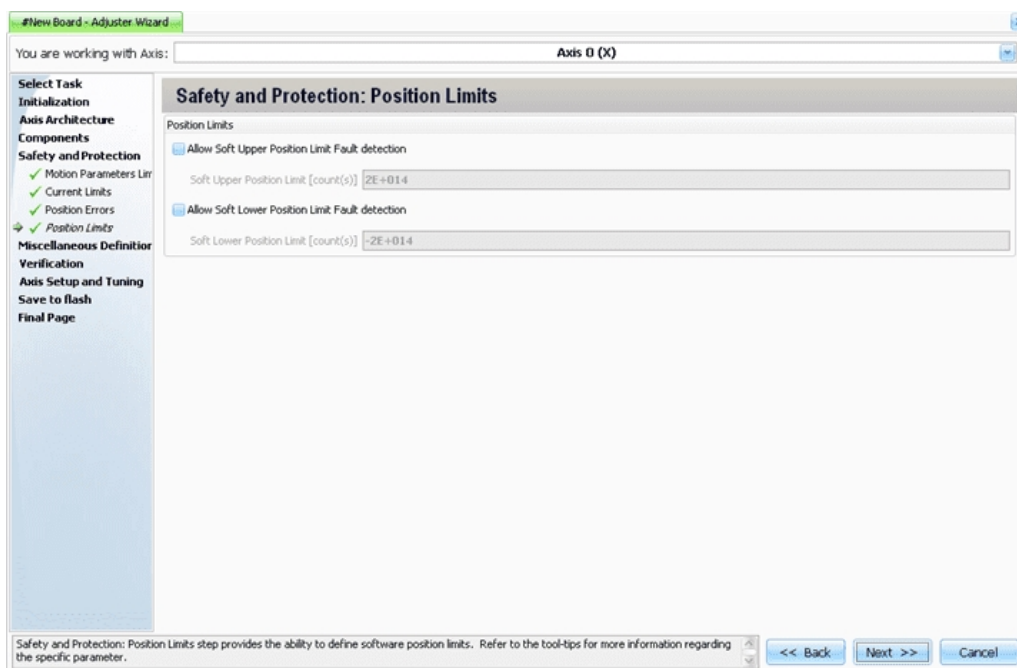


It is recommended to assign high following error fault thresholds for the duration of the configuration and adjustment process. This will accommodate the process while maintaining a measure of safety. Once the process is completed, the thresholds should be reduced to minimal levels.

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.

- > Moving-to-Idle Transition Delay (**DELI**) Time (milliseconds) to wait on transition from acceleration state to idle state.
- > Accelerating-to-Moving Transition Delay (**DELV**) Time (milliseconds) to wait on transition from acceleration to constant velocity state.

6.8.4 Position Limits



This window defines Software Position Limits and enables you to select:

- > **Allow Soft Upper Position Detection (SRLIMIT)** - If Software Right Limit Mask is selected, then the controller examines the fault and kills the motion that is beyond the defined limit.
- > **Allow Soft Lower Position Detection (SLLIMIT)** - If Software Left Limit Mask is selected, then the controller examines the fault and kills the motion that is beyond the defined limit.

6.9 Miscellaneous Definitions

The Miscellaneous Definitions step enables you to set certain variables. There are a total of four possible types of variables that you can set.



The actual number of parameter types and parameters available to you depends on what you defined in the **Axis Architecture** and **Components** steps.

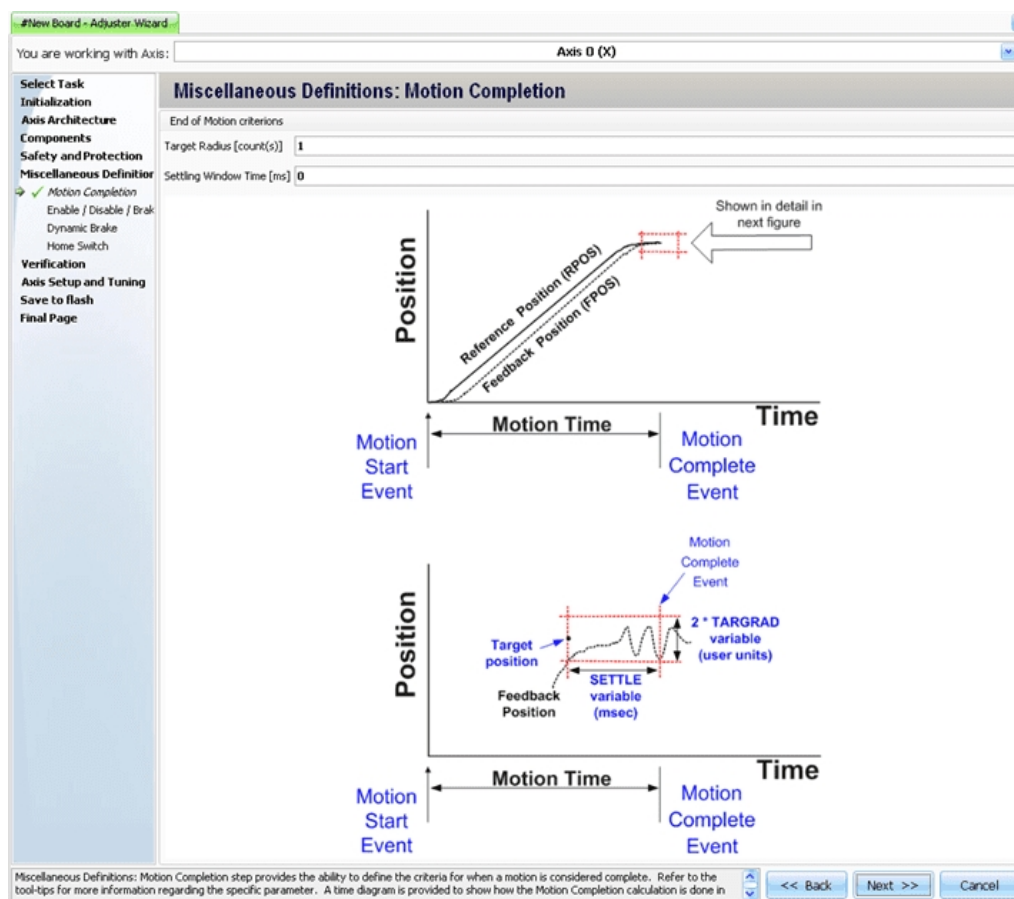
The **Adjuster** displays values it has calculated in the window of each type. You have the option of changing these within the limits of valid values. For details of the variables and their permissible values see the *SPiiPlus ACSPL+ Command & Variable Reference Guide*.

6.9.1 Motion Completion

There are two variables, both optional, for which you can enter values:

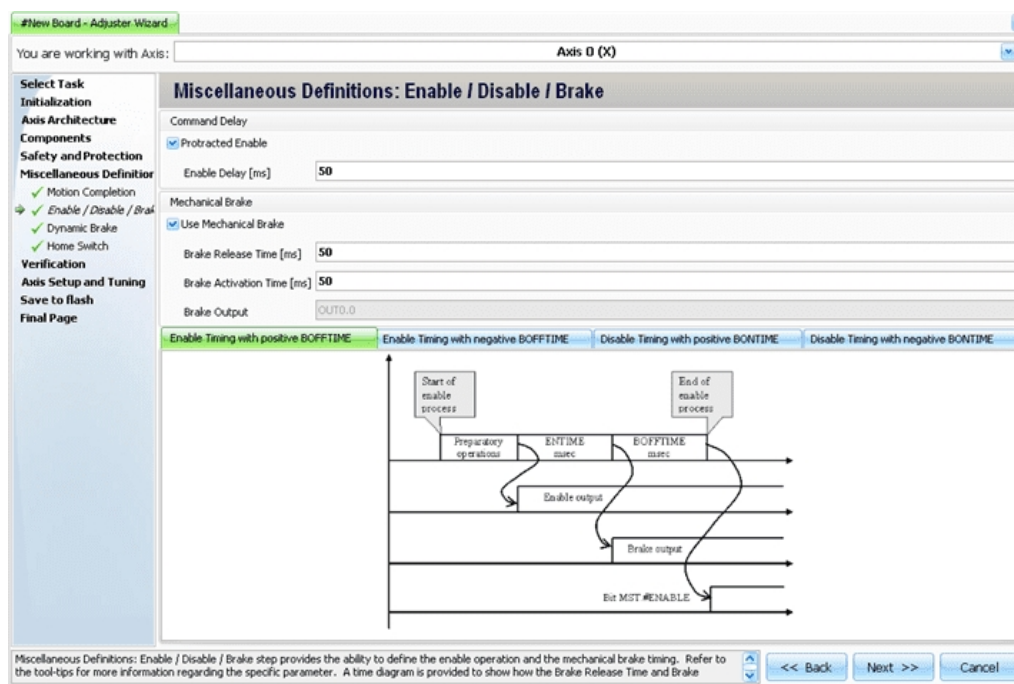
- > Target Radius (**TARGRAD**)
- > Settling Time (**SETTLE**)

To aid your understanding a diagram is included showing the meaning of the two variables.



6.9.2 Enable/Disable/Brake

This window is used for defining the Enable operation and Mechanical Brake timing.



The variables you can define are:

- > **Protracted Enable** - This is optional, but if you select it, you must put a value in **Enable Delay** (ENTIME). The controller delays the Enable GUI by **ENTIME** mSec.
- > **Use Mechanical Brake** - You need to select this only if the application has a mechanical brake; otherwise leave it unselected. If you select it, you have to enter values for:
 - > **Brake Release Time (BOFFTIME)** - This value can be positive or negative and sets when the brake is released.
 - > **Brake Activation Time (BONTIME)** - This value can be positive or negative and sets when the brake is applied.



You are also informed of the Digital Output variable (**OUTx.x**) that will contain the state of brake. Make sure that you include it in your program to test the brake status.

The Digital Output variables for the SPIIPlus family of products are given in the following tables.

Table 5-1. Digital Outputs for Brake Control in SPIIPlus

Digital Output	Controls Mechanical Brake for Axis
OUT1.0	0
OUT1.1	1
OUT1.4	4
OUT1.5	5

Table 5-2. Digital Outputs for Brake Control in MC4U

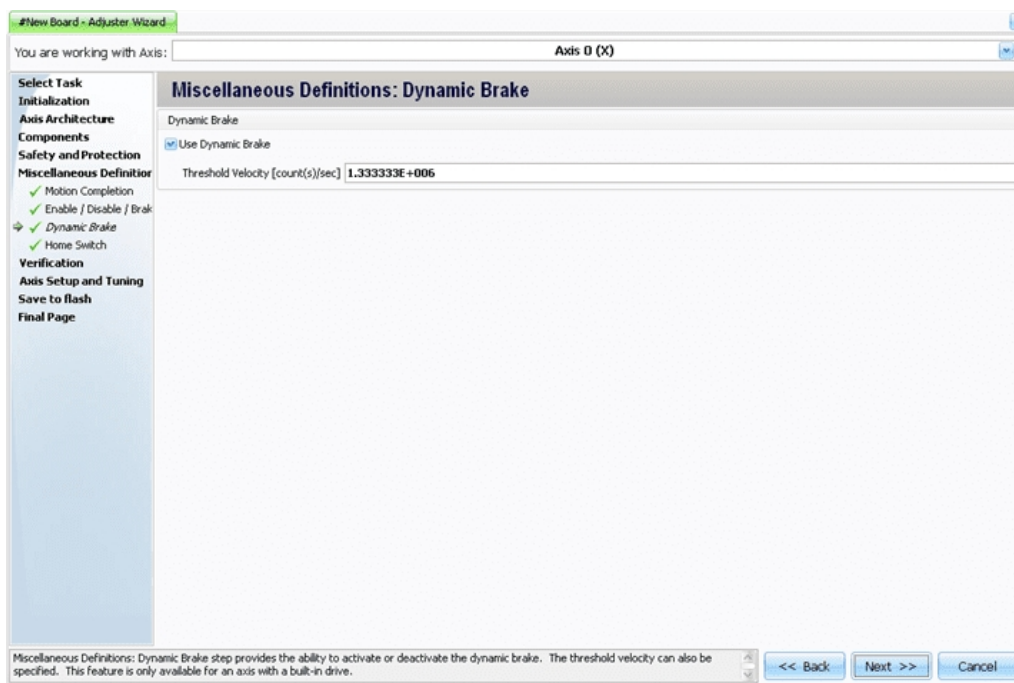
Digital Output	Controls Mechanical Brake for Axis
OUT0.0	0
OUT0.1	1
OUT0.2	4
OUT0..	5
OUT0.4	2
OUT0.5	3
OUT0.6	6
OUT0.7	7

6.9.3 Dynamic Brake

This window enables you to activate (or deactivate) the dynamic brake.



This option is active only for those axes having a built-in drive.



You activate a dynamic brake by:

- > Selecting **Use Dynamic Brake** (this sets the **MFLAGS#DBRAKE** bit for the axis), and then
- > Entering a **Threshold Velocity (VELBRK)** value.

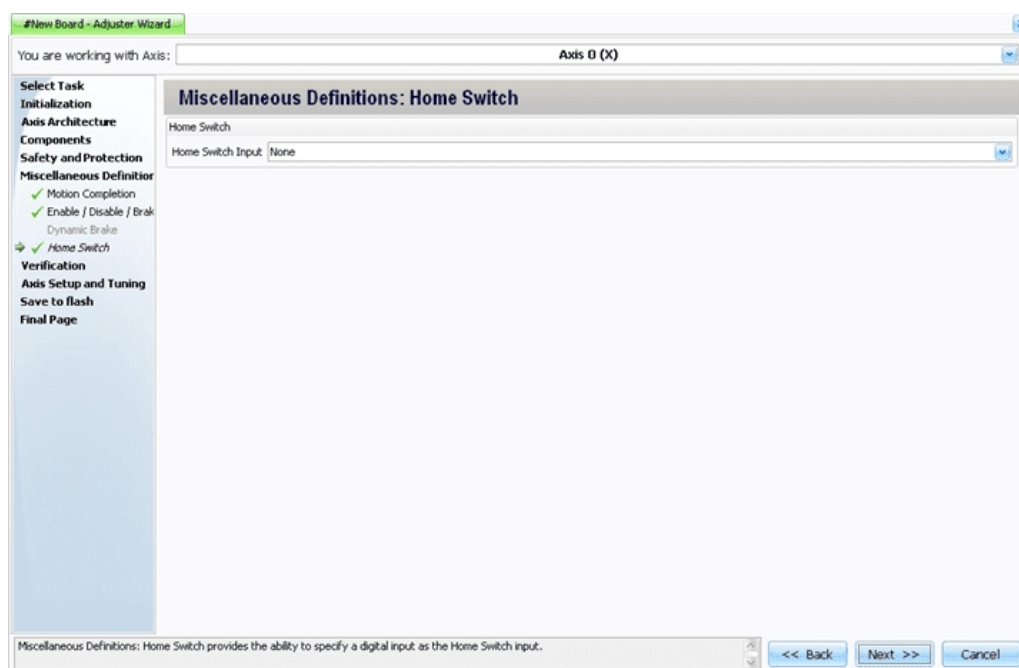
Refer to *SPiiPlus ACSPL+ Command & Variable Reference for details*.

Braking will occur when the following two conditions are met:

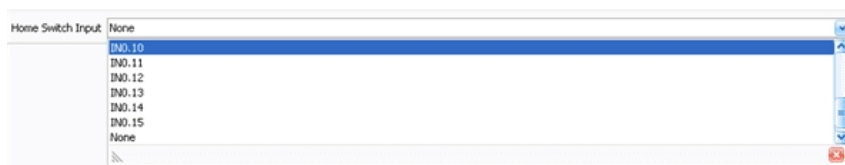
1. The motor is disabled.
2. The feedback velocity (**FVEL**) is less than **VELBRK**.

6.9.4 Home Switch

This window enables you to specify a digital input (**IN**) to signal when the motor is in its home position.



The Home Switch can be **None** or you can select the digital input variable from the **Home Switch** dropdown list:



6.10 Verification






If you have Sin-Cos encoders in the system, before doing this step, perform [Sin Cos Encoder Compensation Window](#) and then return.

The Verification step examines the values you have entered and verifies that they are valid.

Verification is performed on the following:

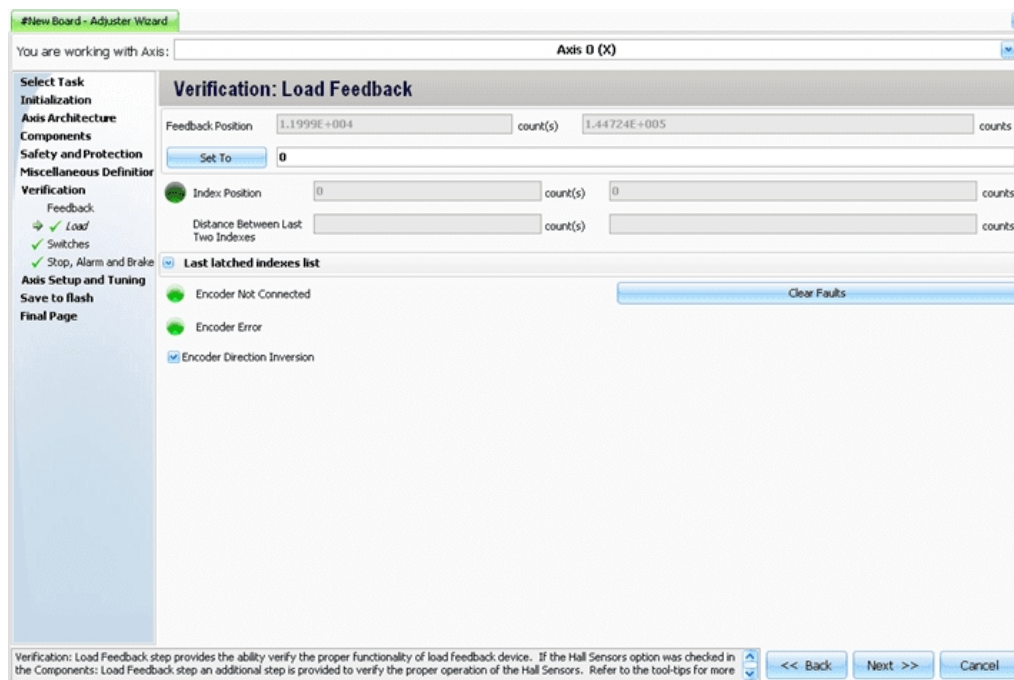
- > Feedback devices
- > Limit switches
- > Hardware Emergency Stop
- > Mechanical Brake

There are three “LED” indicators used throughout the Verification windows:

	Indicates that the item passed verification
	Indicates that the item failed verification
	Indicates that the item is not active

6.10.1 Feedback

This window verifies that the feedback values are valid.



Depending on what you defined in [Components](#), the window displays the values for:

- > Load
- > Motor




If you have defined both Load and Motor as feedback components, there is one page for Load and one page for Motor.

In this window:

- > Check that there are no encoder faults.
- > Check that encoder counts in the right direction. If it does not, click the **Encoder Direction Inversion** check box to reverse the direction
- > Check that the index is working properly by manually moving the motor. The motor should be moved slowly to ensure that each index is latched and displayed.
- > Check that calculated distance between indexes is correct.

To set the feedback (**FPOS**) to a particular value:

1. Enter a value in the **Set To** field.
2. Click  - this resets the position and the **Adjuster** starts counting from this position.

You can view a list of the last latched indexes by clicking the down arrow of the **Last latched indexes list**. To clear this list click **Clear List**.

There are two LEDs in this window relevant to the encoder:

- > **Encoder Not Connected Fault** (FAULT.#ENCNC) - green indicates that this fault was not detected.
- > **Encoder Error Fault** (FAULT.#ENC) - green indicates that no encoder fault was detected.

If you selected **Hall Sensors** in **Components: Feedback**, the following is displayed:



The connection sequence of the three Hall Sensors is not important. You only have to verify that the three Hall sensors are connected and the Hall counter counts 0,1,2,3,4,5 or vice versa. It does not matter if the Hall counters count opposite to the encoder. This will be identified and taken care of during .

The diagram shows:

- > Feedback Position (FPOS) - the position in user units.
- > Hall State - the Hall State transitions: 0^a1^a2^a3^a4^a5 or 5^a4^a3^a2^a1^a0.

The history of the Hall State transitions is displayed under the diagram.


To test the Hall Sensors:

Rotate the motor by hand in one direction. The diagram should show the expected Hall state transitions.

Rotate the motor by hand in opposite direction. The diagram should show the expected Hall state transitions.

Correcting Feedback Values that Fail Verification

Where feedback values fail verification:

1. Return to the **Components:Feedback** step by clicking **Components** in the Task List, then click **Feedback**. The **Components:Feedback** window is displayed.
2. Click **Edit** () and enter a new value or values.



Those values that can be changed are displayed in bold face. Values that cannot be changed are grayed out.

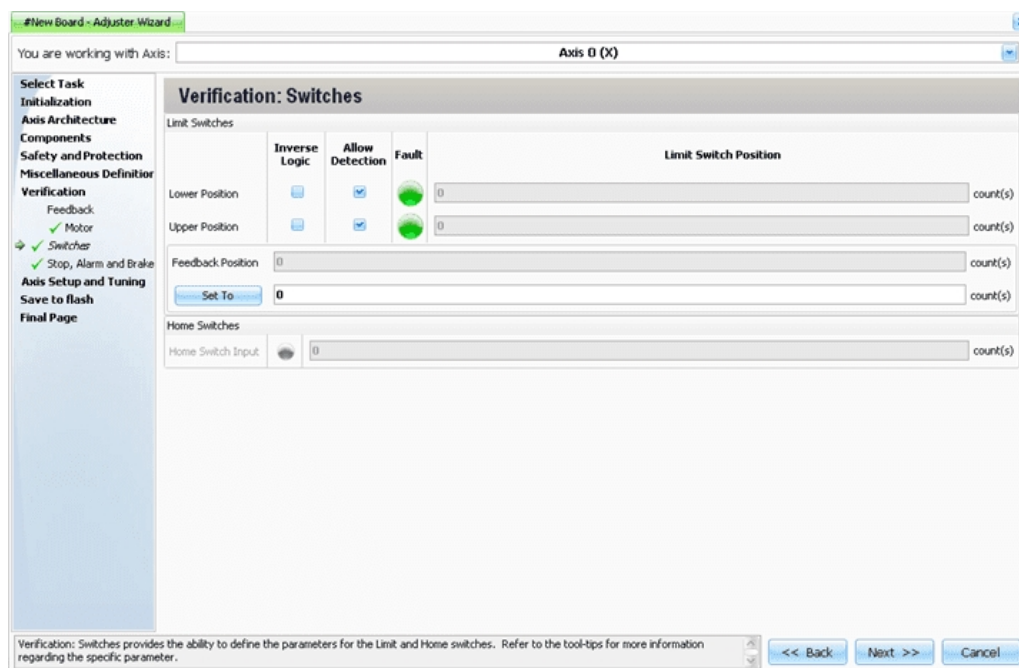
3. Click **Update in Database** and click **Next**. The **Components:Calculate Parameters** window is displayed.



4. Click **Calculate Parameters** then click **Apply changes**. **Adjuster** recalculates the values of the low-level controller parameters and applies them to the appropriate variables.
5. Click **Verification** in the Task List to return to **Verification** and check if the new value(s) pass verification.

6.10.2 Switches

This window is used for verifying the that actual limit switches are properly connected.



Verification of Switch Operability

The LEDs indicate if the switches are properly connected or not. The switches are:

- > **Upper Position Limit (SAFINI.#RL)**
- > **Lower Position Limit (SAFIN.#LL)**

that you set in **Safety and Protection:Position Limits**.



If you have selected a digital input variable in **MISCELLANEOUS DEFINITIONS:HOME SWITCH**, its value is also displayed.

To verify that the switches are properly connected manually move the motor into the switch and check if the Fault LED turns red.

If it does not turn red, this means that there is something wrong with the switch. Check its connection and if it is operational.

Fault Detection Settings

You have the options of:

- > Enable/disable fault detection (**FMASK.#RL** for upper limit, **FMASK.#LL** for lower limit) by selecting the appropriate check box.




If the bit is empty, the controller does not detect the fault.

- > Inverting the detection logic (**SAFINI.#RL** for upper limit and **SAFINI.#LL** for lower limit) by selecting the appropriate check box.

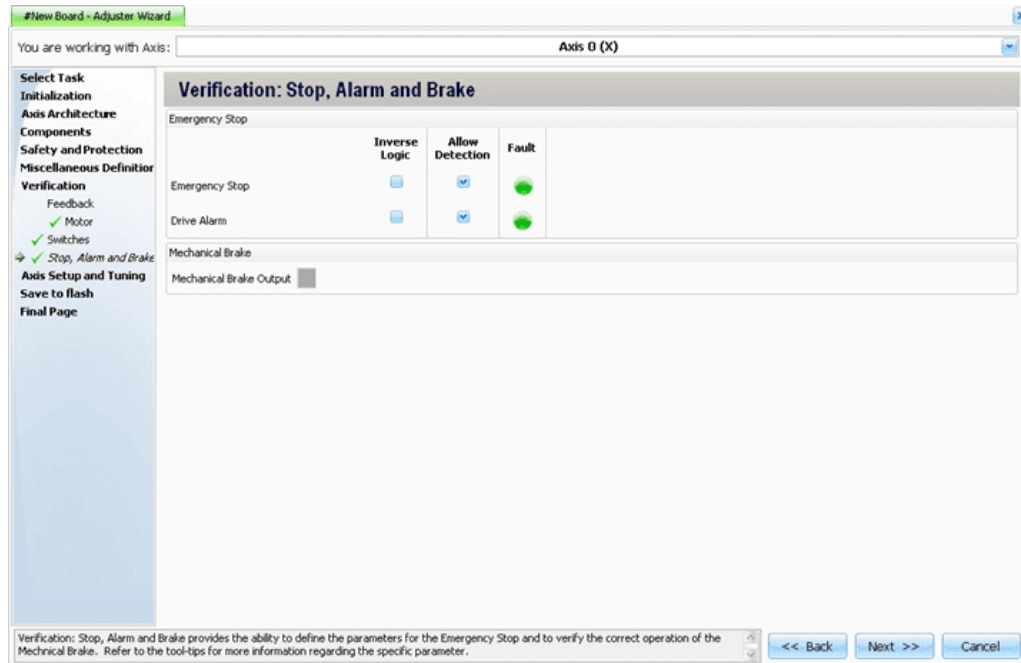
Feedback Setting

To set the feedback (**FPOS**) to a particular value:

1. Enter the value in the **Set To** field.
2. Click  - this resets the position and the Adjuster starts counting from this position.

6.10.3 Stop, Alarm and Brake

This window verifies the status of Hardware Emergency Stop and Mechanical Brake.



The window shows the status of the:

- > Hardware Emergency Stop (**S_SAFIN.#ES**)
- > Drive Fault / Laser Fault (**SAFIN.#DRIVE**)

You have the options of:

- > Enable/disable fault detection (**S_FMASK.#ES** for Hardware Emergency Stop and **S_FMASK.#DRIVE** for Drive Fault / Laser Fault) by selecting the appropriate checkbox.




If the bit is empty, the controller does not detect the fault.

- > Inverting the detection logic (**S_SAFINI.#ES** for Hardware Emergency Stop and **SAFINI.#DRIVE** for Drive Fault / Laser Fault) by selecting the appropriate checkbox.

To set the feedback (**FPOS**) to a particular value:

1. Enter the value in the **Set To** field.

2. Click  - this resets the position and the **Adjuster Wizard** starts counting from this position.
If you defined a Mechanical Brake in **Miscellaneous Definitions:Enable/Disable/Brake**, then you have the option in this window of defining the mechanical brake output by activating it manually, that is, without an ENABLE command by clicking the associated box.
3. When you are done, click **Next** to go to the next window.

6.11 Axis Setup and Tuning

The Axis Setup and Tuning step enables you to fine tune your system. There are a total of six possible options:



The options that are available to you depend on the definitions you entered in **Components**.

- > **Current Loop**
Current Loop enables you to adjust the current loop. Adjustments are done by entering values in the required fields and viewing their effect on the Scope. This is covered in [Current Loop and Current Phase Offset Setup](#).
- > **Current Phase Offset**
Current Phase Offset enables you to verify, and, if needed, adjust the values of the current phase offset. This is covered in [Current Loop and Current Phase Offset Setup](#).
- > **Commutation**
Commutation enables you to adjust motor commutation and generate a commutation startup program.



Commutation applies only for DC brushless (AC Servo) motors. It will appear in the Adjuster tasks only if motors of this type have been defined as part of the system.

This is covered in [Commutation](#).

- > **Open Loop Verification**
Open Loop Verification enables you to check if the motor responds correctly to the drive command, that is, the motor velocity should be in the same direction as drive command. This is covered in [Open Loop and Position Verification](#).
- > **Position and Velocity Loops**
Position and Velocity enables you to tune the position and velocity loops in order to optimize the servo system performance. This is covered in [Position and Velocity Loops](#).
- > **Position Verification**
Position Verification is relevant only for open loop modes and enables you to verify that the actual move corresponds to a commanded one, and make changes where necessary. This is covered in [Open Loop and Position Verification](#).

7. Current Loop and Current Phase Offset Setup

This chapter covers Current Loop and Current Phase setup and tuning procedures of the SPiiPlus MMI Application Studio **Adjuster Wizard** Axis Setup and Tuning step of the Setup New System or Controller Task.

7.1 Current Loop

Current Loop enables you to adjust the current loop.

Current Loop adjustment applies only to:



- > PWM, Digital Current Control, controller-drive interface with:
 - > DC brushless/AC servo motor (two or three phase)
 - > Step motor (with feedback, close loop)
 - > DC brush or single phase motor
 - > AC induction motor
- > Analog ($\pm 10V$), Digital Current Control, controller-drive interface with
 - > Three phase DC brushless/AC servo motor
 - > DC brush or single phase

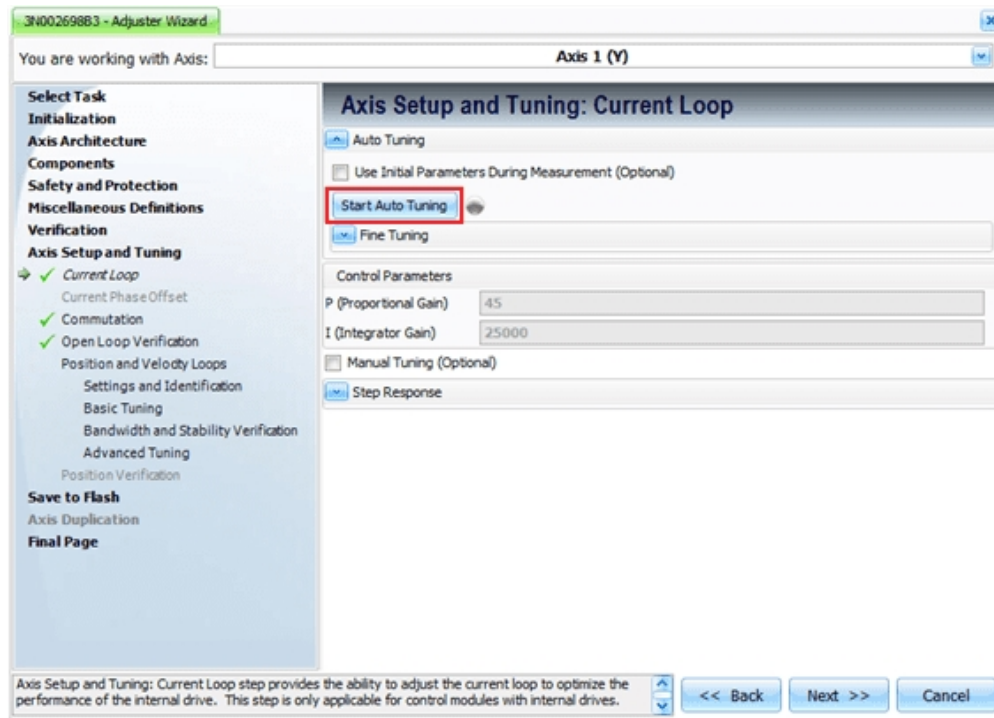
Two current loop tuning methods are available:

- > Auto tuning - The auto tuning process calculates the current loop parameters from values measured by the frequency response.
- > Manual tuning - The manual tuning process calculates the current loop parameters from a step response with parameters you set. Manual tuning is optional.

Auto tuning

The calculated current loop parameters are:

- > The Proportional Gain (SLKP)
 - > The Integrator Gain (SLKI)
1. To start the auto tuning process, click **Start Auto Tuning**.



- > When the auto tuning process starts, the **Start Auto Tuning** button changes to **Stop Auto Tuning** and the LED next to it flashes "green".
- > When the auto tuning process is complete, the LED returns to "gray" and **Done** shows to the right of it.

2. When the auto tuning is complete, click **Next** to go to the next adjustment.

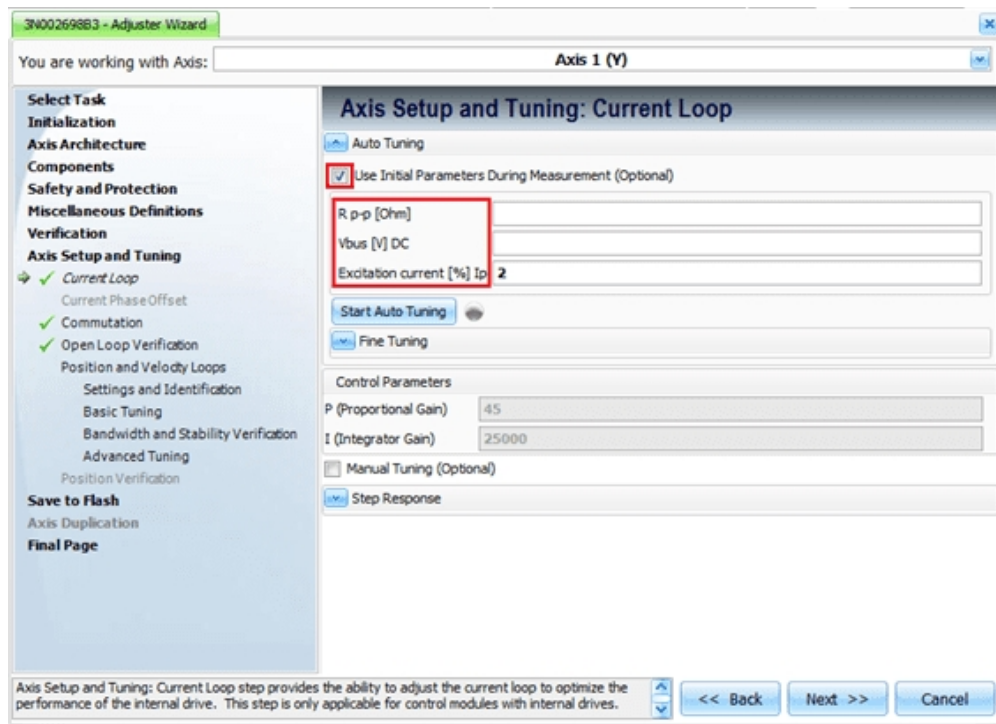
Auto tuning with Use Initial Parameters During Measurement (Optional)

If necessary, the current loop auto tuning can be done from initial parameters that you set. The parameters are:

- > Motor resistance phase-to-phase
- > The rectified bus voltage
- > Initial excitation current

To start the Auto Tuning process with the optional **Use Initial Parameters During Measurement**:

1. Click **Use Initial Parameters During Measurement (Optional)** check box.
2. Enter the values in the applicable field.
3. Select **Start Auto Tuning**.

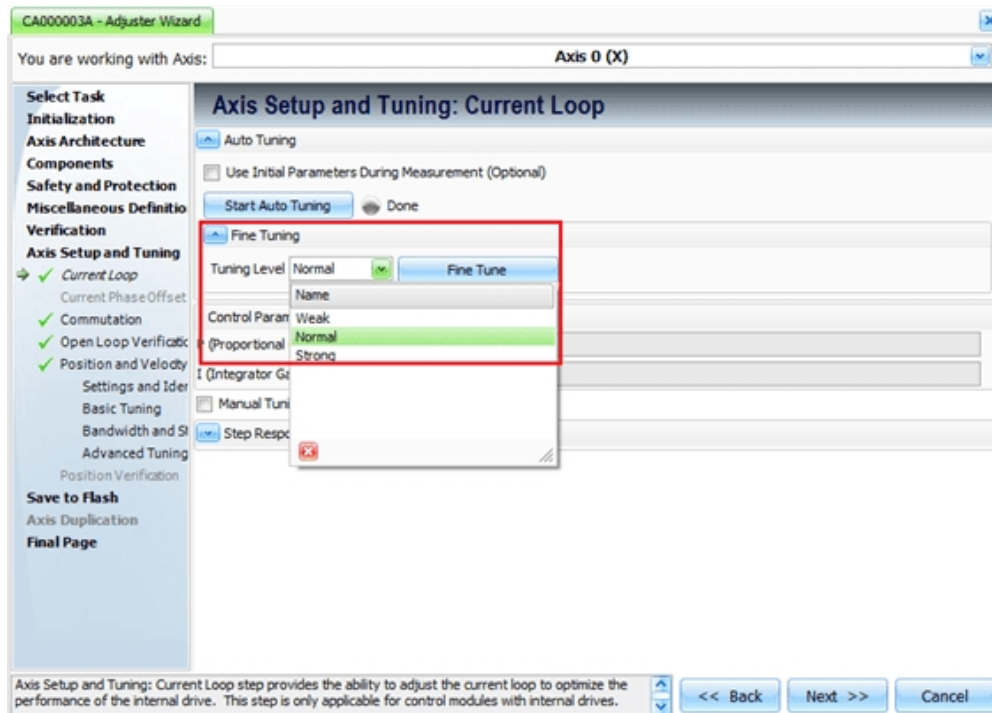


4. When the auto tuning is complete, click **Next** to go to the next adjustment.

Fine Tuning

When Auto tuning is complete, a **Fine Tuning** process can be done. Auto fine tuning is only for PWM drives with good linear properties. You can do fine tuning for three different types of drives :

- > Weak (PWM drives with poor linear properties)
 - > Normal (PWM drives with good linear properties)
 - > Strong (PWM drives with good linear properties)
1. To start fine tuning, click **Fine Tuning** drop down icon to show the different tuning levels.
 2. Select the applicable drive.
 3. Click **Fine Tune**.

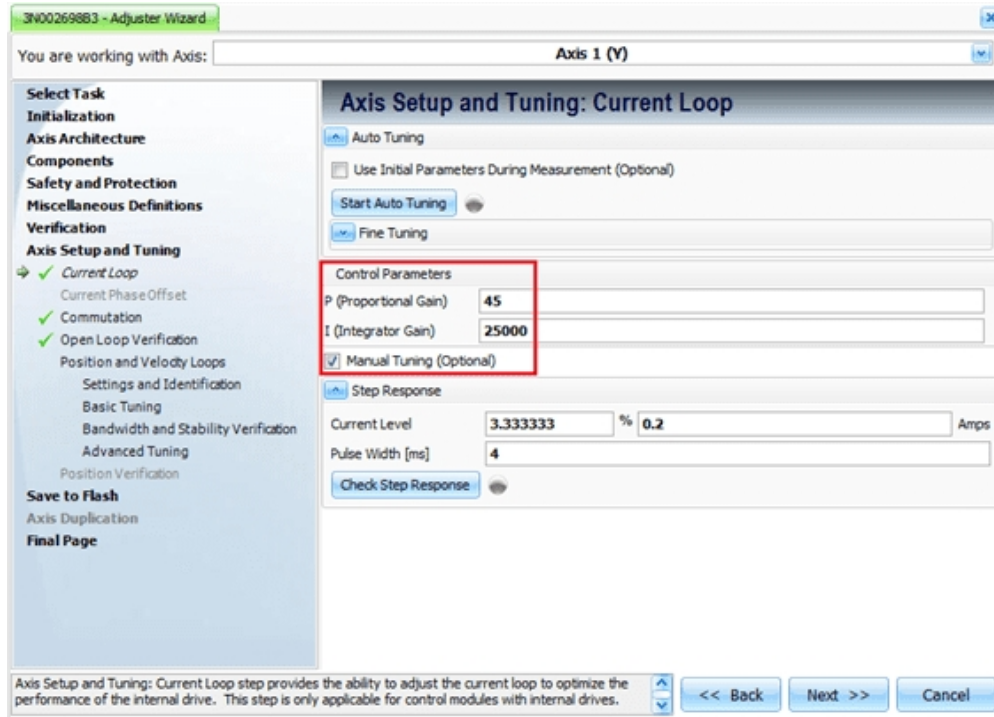


4. When the fine tuning is complete, click **Next** to go to the next adjustment.

Manual Tuning (Optional)

If you select manual tuning, you can set the proportional gain (SLKP) and the integrator gain (SLKI).

1. Click the **Manual Tuning** check box.
2. Enter the values in the applicable field.

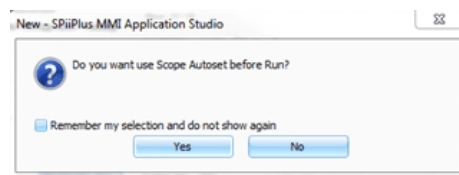


3. Click **Next** to go to the next adjustment.

Step Response

Use step response to verify the proportional gain and integrator gain. The required parameters to do a step response are:

- > Current level
 - > Pulse width
1. To start the step response, click the **Step Response** drop down box to show the parameter fields available.
 2. Enter the parameter values.
 3. Click **Check Step Response**. A dialog box opens and shows the message.



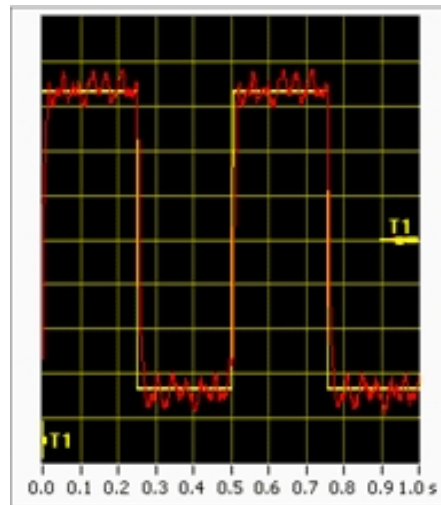
4. Select the applicable condition.



When the step response starts, the **Check Step Response** button changes to **Stop** and the LED next to it flashes "green".

5. To make adjustments on the signal:

- > Increase the Proportional Gain (SLKP) until the response waveform approximates a square and a small overshoot appears, for example:



- > Increase the Integrator Gain (SLKI) by hundreds, until you get a narrow overshoot of 10% to 20%.
6. When you achieve the optimal values, click **Stop**. Close the Scope.
 7. Click **Next** to go to the next adjustment.

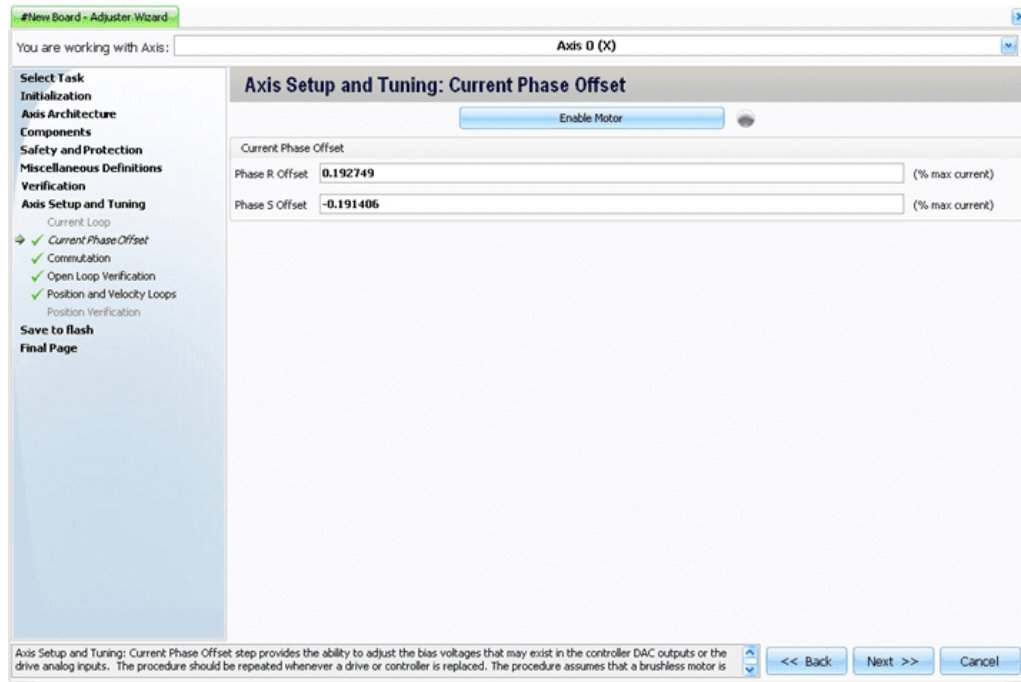
7.2 Current Phase Offset

The Current Phase Offset window enables you to verify, and, if needed, adjust the values of the current phase offset.

Current Phase Offset applies to Analog ($\pm 10V$) controller-drive interfaces with two or 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. If, however, 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.



There are two current phase offset variables:

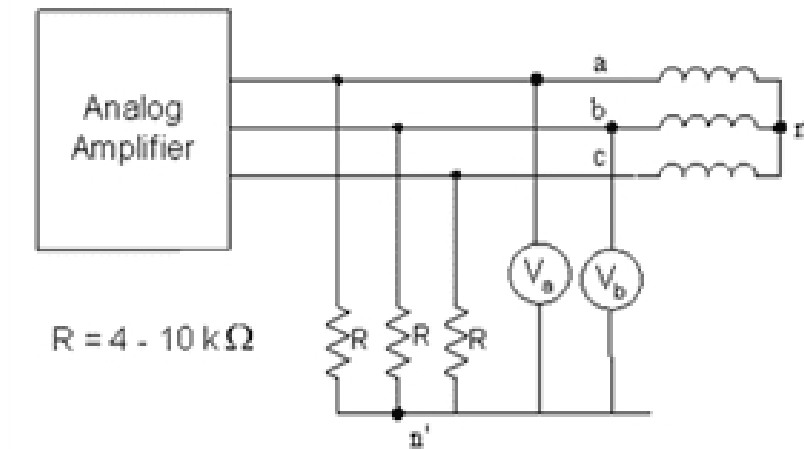
- > **Phase R Offset (SLBIASA)** which contains the maximum controller voltage output (expressed as a percentage).
 For integrated models: **SLBIASA** specifies the measured value of the current input bias of Phase R.
 For nonintegrated models: **SLBIASA** specifies the bias of the drive output. The controller uses the value only for brushless motors commutated by the controller.
- > **Phase S Offset (SLBIASB)** which is used for defining the maximum controller current (expressed as a percentage).
 For integrated models: **SLBIASB** specifies the measured value of the current input bias of Phase S.
 For nonintegrated models: **SLBIASB** specifies the bias of the drive output. The controller uses the value only for brushless motors commutated by the controller.

Click **Enable Motor** and check that the LED turns green.

If the LED turns red, this means that one or both values need to be adjusted, in which case, 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:

1. Make sure that the value of the **DOUT** variable is zero (you can query it from the **Communication Terminal**, for example: `?X_DOUT`).
2. Connect the voltmeter between phase "R" and the motor neutral point, and adjust **SLBIASA** to make the reading as close to zero as possible. A typical range is $\pm 10\%$. The value can be either positive or negative.
3. Connect the voltmeter between phase "S" and the motor neutral point, and adjust **SLBIASB** to make the reading as close to zero as possible.

If the neutral point is not accessible, or the motor is connected in delta, you 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 of 4-10 k Ω . Then you should measure relative to the artificial neutral as shown in the following drawing:



When you are done, click **Next** to go to the next window.

8. Commutation



Commutation applies only for DC brushless (AC Servo) motors. It will appear in the **Adjuster** tasks only if motors of this type have been defined as part of the system.

8.1 Adjuster and Commutation

8.1.1 Commutation 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, on the other hand, 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.

SPiiPlus motion controllers support three types of drives for 3-phase DC brushless/AC servo motors:

- > **Universal Internal Drive** (such as SPiiPlus CM and MC4U drives). Commutation is performed by the controller as part of its Field-Oriented Control algorithm.
- > **Two-Input Drive** (such as SPiiPlus SA controller or SPiiPlus PCI card with a Sine Wave Brushless Amplifier). The drive receives two current commands from the controller, reflecting required current for two of the motor phases. In this case **the controller is responsible for commutation**.
- > **One-Input External Drive** (such as a SPiiPlus SA controller with an Analog Input Brushless Amplifier). This drive receives a single current command from the controller ($\pm 10V$), reflecting the required current amplitude, and takes care of the commutation by itself.



Commutation needs to be done only for types 1 and 2.

8.1.2 Commutation Technical Details

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

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

$$I_B = I * \cos(\phi + 120^\circ)$$

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:

$$\phi = k * CP + \Theta.$$

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 Adjuster calculates **k** according to the motor and encoder variables that you specified in [Axis Setup and Tuning](#).

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 Θ_0 . A position **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:

$$\Theta = \Theta_0 - K * CP_0 + 90^\circ$$

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 depends on the motor position, the commutation process has to be done only once for an absolute encoder but after every power-up for incremental quadrature and Sin/Cos encoders – 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. For this reason you can use **Adjuster** to generate and store a **Commutation Startup Program**.

8.1.3 The Adjuster Commutation Task

The **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 both of the motor and the encoder.
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.

The **Commutation Startup Program** is an ACSPL+ program that is used to retrieve commutation. It can be executed upon 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 see *SPiiPlus ACSPL+ Command & Variable Reference Guide*), which is fast commutation retrieval using a closed-loop algorithm (and involving almost no motor movement).

8.2 Performing the Commutation Task



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.



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.



When the page is first displayed, Preferences and Advanced Parameters are not displayed. It is strongly recommended that you expand both and view all the options available to you.

8.2.1 Preferences



When initially setting **Preferences**, you should first use the default values (by clicking **Default**). This will set the commutation parameters in accordance with the settings you have entered in previous steps of the Adjuster.

Excitation Current	<p>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: $0.95 \times \mathbf{XRMS}$, \mathbf{XRMSD}, or \mathbf{XRMSM}.</p> <p>Things you should consider:</p> <ul style="list-style-type: none"> > 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. > 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. <p>It is recommended starting the commutation process with an excitation current of $0.4 \times \mathbf{XRMS}$, \mathbf{XRMSD}, or \mathbf{XRMSM}. If the results are not successful, repeat this task with higher excitation current values.</p>
Search Velocity	<p>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.</p>
Settling Window Time	<p>Determines the time assigned to the motor for settling at detent points.</p>
Retrieve Commutation Phase at	<p>In all the schemes except the powerup position scheme, the following steps are performed:</p> <ul style="list-style-type: none"> > Settle at a first detent point. > Move the current vector to identify the phase sequence. Automatically correct for wrong polarity. > 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. > Retrieve the commutation phase according to the selected scheme. <p>Considerations when selecting a Commutation Scheme:</p> <ol style="list-style-type: none"> 1. Based on detent point: 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. 2. Based on index: Saves the value of the commutation phase at the index position. This value can be used at subsequent system

	<p>powerups to achieve the same commutation result. Six variations of this scheme are available:</p> <ul style="list-style-type: none"> > First Index in Positive Direction > First Index Next to Right Hard Stop > First Index Next to Right Limit Switch > First Index in Negative Direction > First Index Next to Left Hard Stop > First Index Next to Left Limit Switch <p>3. Other</p> <ul style="list-style-type: none"> > Powerup Position > Hall Signals Transition > Current Absolute Position
--	---

8.2.2 Advanced Parameters

Initial Commutation Offset	<p>Determines the initial value of the commutation phase, and, thereby, the initial orientation of the current vector and the initial detent point.</p> <p>By properly setting this offset, you can avoid an initial jump in a system with a predefined startup position. For other systems it can be used to determine a specific target position for the initial jump.</p> <p>If no information is known about the relationship between the current vector and the motor position, it should be set to 0 (the default).</p>
Maximum Search Distance	<p>Sets the maximum allowed distance for searching for targets (index or limits). It is recommended to set the variable according to the operational distance.</p>
Verify Control and Feedback Parameters	<p>Causes the system verify the encoder and motor variables during the initial commutation 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.</p>
Display Detailed Commutation Results	<p>Displays commutation phase data (in electrical degrees).</p>

Use Hall Signals for Continuous Commutation*	This option resolves a commutation loss due to improper incremental encoder-to-motor rotation counts. When this option is selected, the commutation angle is adjusted at each hall transition. In between hall transitions, a sinusoidal commutation is calculated based on incremental feedback.
Calculated Hall Alignment Offset*	Compensates for Improper hall alignment and optimizes torque/force production.
Number of Magnetic Pitches to Measure	A word of 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.

*Only available when Hall Signals Transition is the selected option for "Retrieve Commutation Phase at". This is the default condition if Hall sensors were selected.



You are working with Axis: **Axis 0**

Axis Setup and Tuning: Commutation

Commutation Program: **Standard** Run in Buffer: **0** Default

Preferences

Excitation Current: **47.5** % of maximum

Search Velocity: **2000** count(s)/sec

Settling Window Time: **1000** ms

Retrieve Commutation Phase at: **Hall Signals Transition**

Advanced Parameters

Initial Commutation Offset: **0** degrees

Maximum Search Distance: **Two Rotations**

Verify Control and Feedback Parameters: ☒

Display Detailed Commutation Results: ☐

Use Hall Signals for Continuous Commutation: ☐

Calculate Hall Alignment Offset: ☐

Number of Magnetic Pitches to Measure: **4**

Make sure that the unsolicited messages are not shown in Communication Terminal, otherwise the commutation program output will be partially shown in Communication Terminal and partially in commutation program output window.

Axis Setup and Tuning: Commutation step provides the ability to perform a motor commutation and generate a start-up program. This step only applies to motors that have commutation performed by the

<< Back Next >> Cancel

8.2.3 Commutation Startup Program

Commutation Startup Program

The options that you have when setting the Commutation Startup program are illustrated in the following diagram:

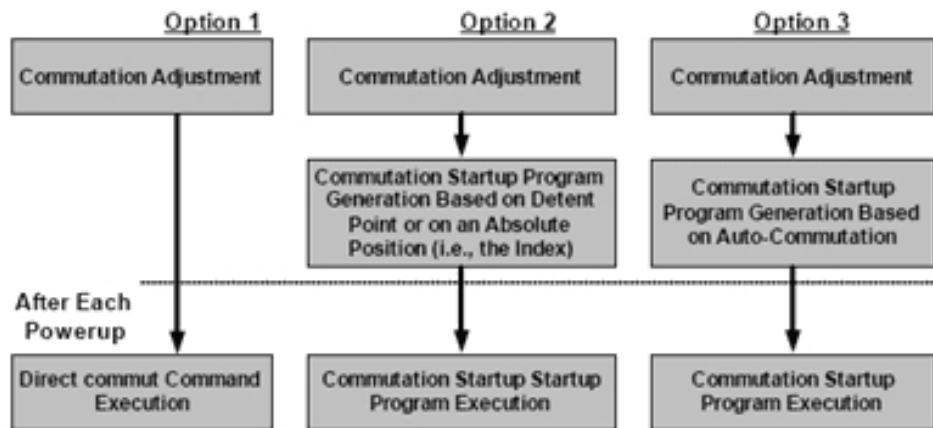
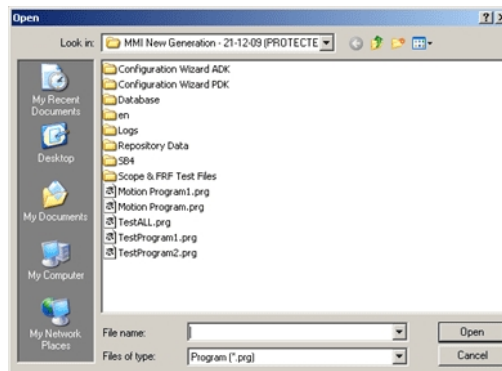


Figure 7-1. Commutation Options

To set the Commutation Startup program:

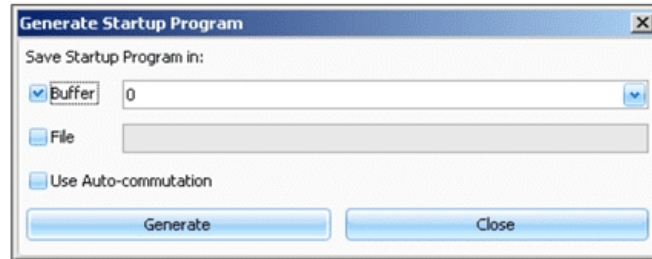
1. Select one of the two Commutation Program options from the **Commutation Program** dropdown list:
 - > **Standard**: Uses a built-in default commutation scheme that is incorporated in **Adjuster**.
 - > **User Defined**: For selecting a commutation program that was previously saved in a file (in this case you need to have prepared the Commutation Startup program through **Program Manager**. A browser is displayed enabling you to search for and load your program:




2. Select the required buffer for storing the Commutation Startup program from the **Run in Buffer** dropdown list (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.
3. Set your preferences (see [Preferences](#) and [Advanced Parameters](#)), or click **Default** which will enter default values in the preferences.
4. At this point you have two choices:
 - a. **Start Commutation Program** - causes the **Adjuster** to automatically generate the Commutation program taking the **Retrieve**

Commutation Phase at parameter as its starting point. It compiles the program and runs it.

- b. **Generate Startup Program** - causes the **Adjuster** to generate the Commutation Startup program. You are prompted with:



The **Buffer** checkbox should be selected and the buffer number that you entered in Step 2 should appear. If the program is to be generated from a user file, select the **File** checkbox and click  to browse for the file. If you want to use automatic commutation, select the **Use Auto-commutation** checkbox.



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.

Click **Generate**. In the **Program Manager** pod, note that program appears in the specified buffer, and a notification that the program was successfully generated is displayed.



Click **Start Startup Program** to run the Commutation Startup program.

Observe the motion of the motor and the messages that appear in the **Commutation Program Output** panel. If errors appear or the motor does not respond as it should, make adjustments to the parameters and run again - refer to [Troubleshooting Commutation](#).

8.2.4 Commutation Program Output Panel

Commutation Program Output Panel

There are two buttons associated with the **Commutation Program Output** panel:

- >  - clears the messages in panel.
- >  - enables you to save all the messages to a text file.

When you are done, click **Next** to go to the next **Adjuster** task.

8.3 Troubleshooting Commutation

Troubleshooting Commutation

If the motor does not move during **Start Commutation Program** or **Start 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 **Start Commutation Program** or **Start Startup Program** execution displays a message about wrong motor variables, return to [Components](#) 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**, **XRMSD**, or **XRMSM** value. If the motor and drive rating allow it, try to increase the value.
- > There is an obstacle in the path of the motion.

Another problem that can arise is the case where **Verify Control & Feedback Parameters** is not selected in the [Advanced Parameters](#). 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.

9. Open Loop and Position Verification

This chapter covers Open Loop and Position verification procedures of the SPiiPlus MMI Application Studio **Adjuster Wizard** Axis Setup and Tuning step of the Setup New System or Controller Task.



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

9.1 Open Loop Verification

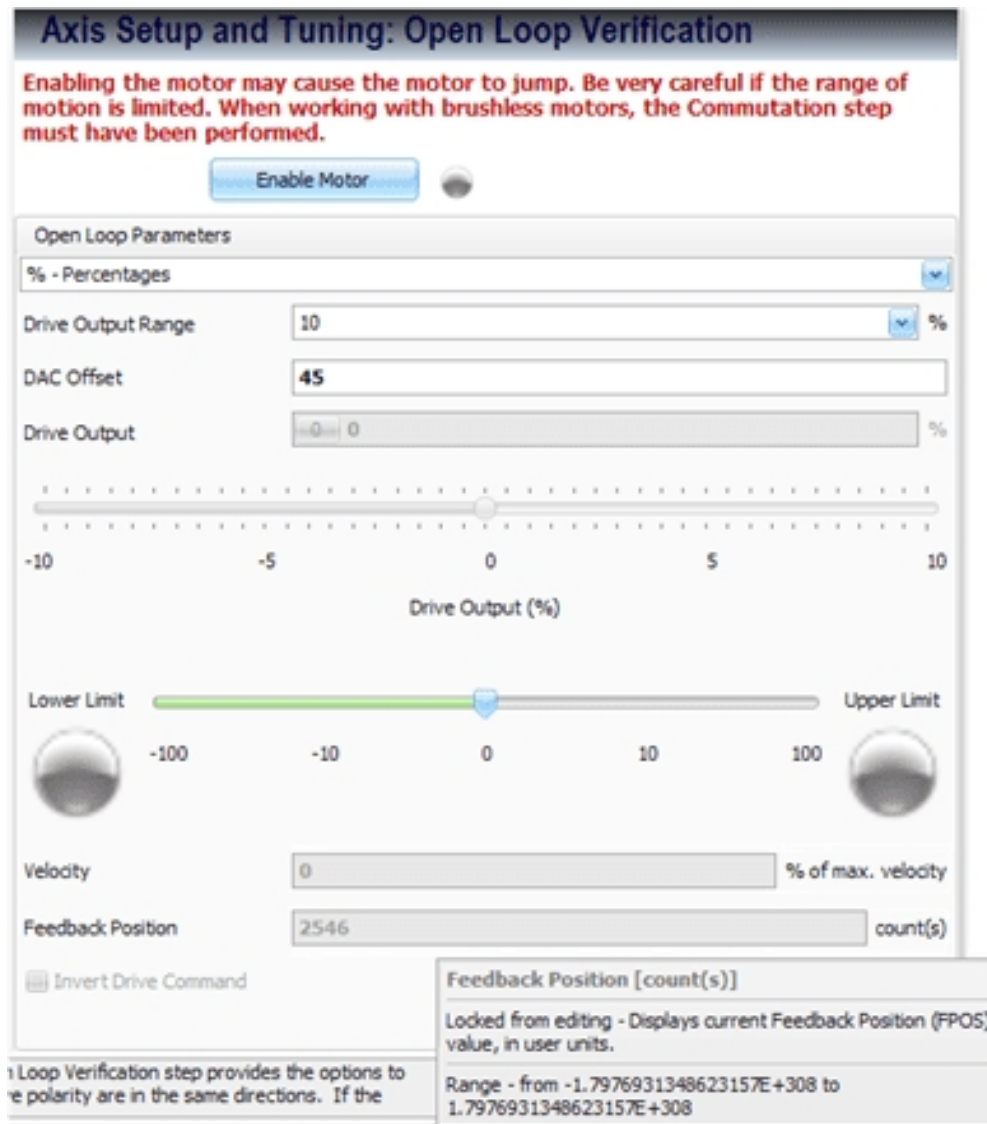
The Open Loop Verification window enables you to check if the motor responds correctly to the drive command, that is, the motor velocity should be in the same direction as drive command.

Enabling the motor may cause it to jump. To avoid personal injury or damage to the equipment, check the following before enabling the motor:



- > Ensure that nothing (people, cables, or other obstacles) is in the path of the motor or objects connected to the motor.
- > Ensure that the motor is securely anchored and that proper safety barriers, stops, and limits are installed.

Be ready to engage the Emergency Stop switch.



Values can be expressed either as:

- > % - Percentages
- > Amps - Amperes

which you select from the dropdown list.

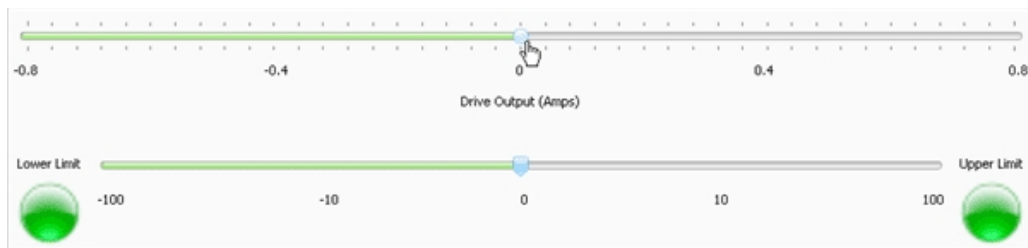
The parameters on this window are:

- > **Drive Output Range** - Percentage (or amperage) relative to the maximum voltage as defined in **Components** - see [Drive](#).
- > **DAC Offset** - Offset to be added to the results of the current loop control. This value is stored in the **SLIOFFS** variable.

The offset can be used to compensate for an active component of the motor load. For example, in a vertical axis the weight of the carriage can be compensated.

1. Click **Enable** to enable the drive.
2. If an encoder is active in the system, click **Zero** to reset the encoder.

3. Drag the Drive Output slider slowly to the right until the motor moves. If the motor does not move (due to friction), increase the **Drive Output Range** (with the slider or by entering a value) until the motor moves.



4. Verify that the velocity gauge indicator moves to the right (same direction you moved the slider). If the velocity arrow moves in the opposite direction, select **Invert Drive Command**.

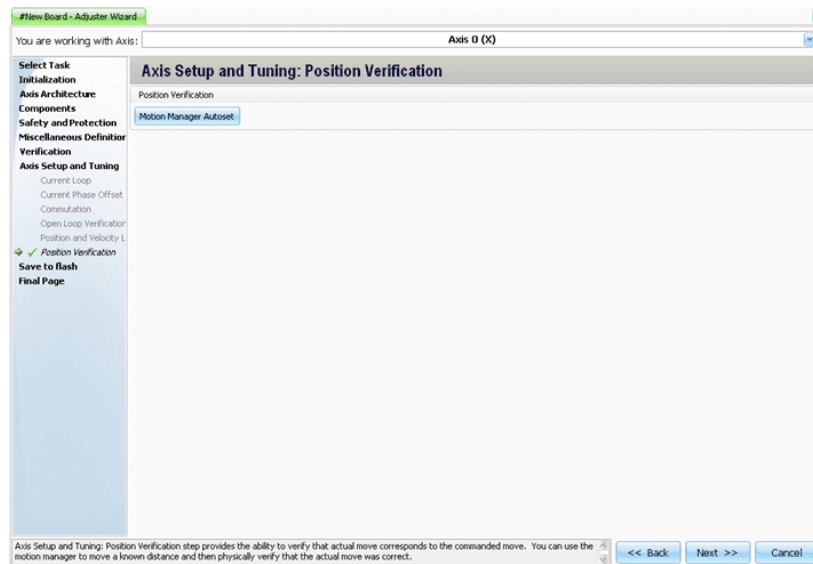


If the direction of the velocity gauge is opposite to that of the Drive Output slider, it is an indication of positive feedback. This is likely to cause a motor run away. If the direction of the velocity gauge is opposite to that of the Drive Output slider, it is an indication of positive feedback. This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

5. 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.

9.2 Position Verification

The Position Verification window is relevant only for open loop modes for step motors and enables you to verify that the actual move corresponds to a commanded one, and make changes where necessary.



To tune position verification:

1. Click **Motion Manager Autoset**. The Motion Manager panel is displayed.



The **Motion Type** dropdown list gives you the option of defining the type of motion, the options are:

- > One Direction Repeated Move
- > Back and Forth Move (Default)

Position is always measured between two points:

- > Point A
- > Point B

Point A is the start position and **Point B** is the end position. The motion is then measured from Point A to Point B.



You can set these points by manually moving the motor to Point A and clicking **Read**, and repeating this for Point B.

You can change the sign of the value by clicking **+/-**

The **Dwell** value is Dwell time between moves.

Feedback Position is the point where the feedback is to be measured. You can reset this position to zero by

clicking **0**

Position Error is a measure of the error in position.

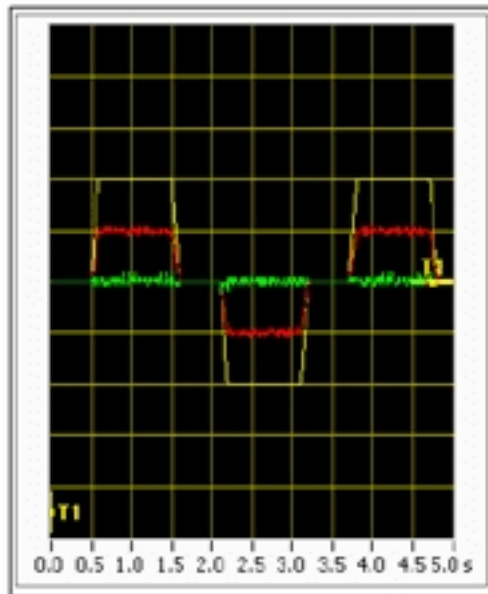
The Motion Parameters that you can manipulate are:

- > **Velocity** (VEL)
- > **Acceleration** (ACC)
- > **Deceleration** (DEC)
- > **Kill Deceleration** (KDEC)
- > **Jerk** (JERK)

To start the measurement:

1. Click **Scope Autoset**, this displays the Scope to measure position.
2. Click **Enable**, this enables the motor of the axis you are working on.
3. Click **Start motion**, this starts the motion.
Visually check the motion. If there is a positional error, exit **Adjuster** and correct the value in your program, then run **Adjuster** again.
4. Check the **Scope** display.

The ideal Scope display is a square wave indicating that the Position is well adjusted.



If the jitter is too extreme, adjust the parameters until the scope shows that the motion graph closely matches the reference lines.

10. Position and Velocity Loops

This chapter covers the Position and Velocity procedure of the SPiiPlus MMI Application Studio **Adjuster Wizard** Axis Setup and Tuning step of the Setup New System or Controller Task.

The Position and Velocity Loops window enables you to tune the position and velocity loops in order to optimize the servo system performance.



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.

The window has two modes:

- > Velocity Loop - for tuning the velocity loop
- > Position Loop - for tuning the position loop

You have to use both modes, one after the other, to completely tune the servo loops. In addition, you make use of two tools:

- > **Scope**, and
- > **Motion Manager**

When you enter the Position and Velocity window, you:

1. Select mode from the **Mode** dropdown list.
2. Click to display the **Motion Manager** panel.
3. Click to display the **Scope**.

The MMI display should look as shown in [Figure 9-1](#)

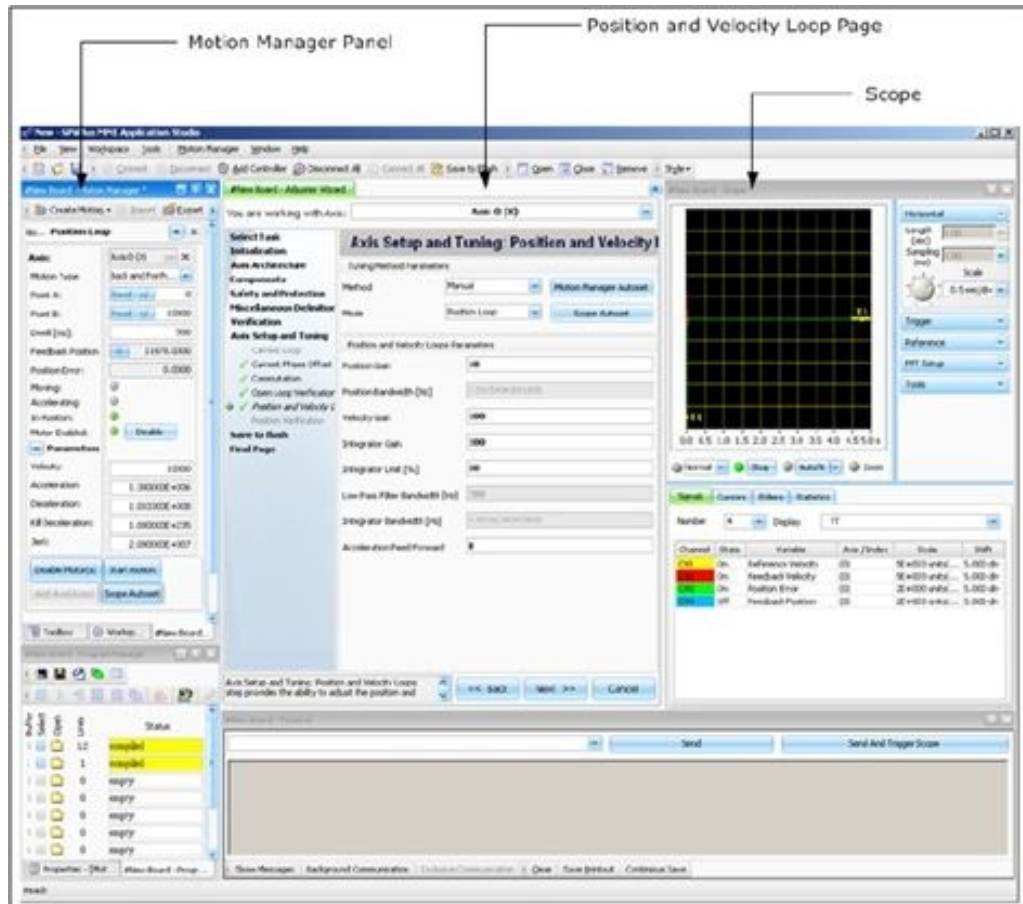


Figure 9-1. Velocity and Position Loop Adjustment - General Layout

10.1 Velocity Loop

The parameters associated with velocity in the Position and Velocity window in the Velocity Loop mode are:

- > **Velocity Gain (SLVKP)** - Proportional coefficient in velocity servo loop.
- > **Integrator Gain (SLVKI)** - Integrator coefficient in velocity servo loop.
- > **Integrator Limit [%] (SLVLI)** - Used to prevent the integrator from exceeding a certain value.



The default value, 50%, is adequate for most systems.

- > **Low Pass Filter Bandwidth (SLVSOF)** - Second order filter bandwidth.
- > **Integrator Bandwidth** - Calculated by **Adjuster**, and is approximately equal to SLVI/20.5.



You can change the values only of those parameters that are not greyed out.

The **Motion Manager** for Velocity Loop adjustments serves for controlling the motion in order to observe it graphically in the **Scope**.

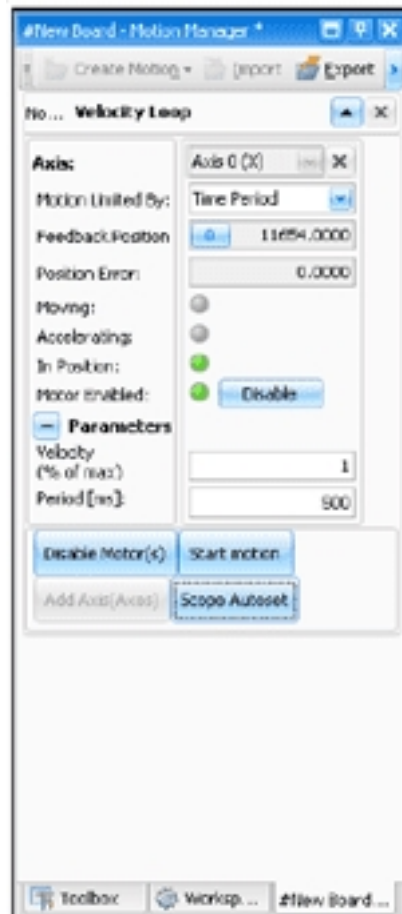


Figure 9-2. Velocity Loop Motion Manager Panel

The **Motion Limited By** dropdown list gives you the option of limiting the measurement to:



- > **Time Period** - Typically used with rotary motion. Period sets the time intervals
- > **Between Two Points** - Typically used with back-and-forth linear motion.

When you select this option, two more fields are displayed:

- > **Point A**
- > **Point B**

Point A is the start position and Point B is the end position. The motion is then measured from Point A to Point B.



Note You can set these points by manually moving the motor to Point A and clicking , and repeating this for Point B. You can change the sign of the value by clicking .

- > Positive Direction Only - Motor moves in the positive direction only. Period sets the time intervals.
- > Negative Direction Only - Motor moves in the negative direction only. Period sets time intervals.



The parameters that are available for changing depend on the type of controller and motor.

Parameters that always appear are

- > **Velocity (% of Max)** - which enables you to slow the motion by a certain percentage.
- > **Period** - which enables you to define the period, in milliseconds.

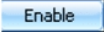
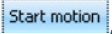
Other parameters that may appear are: For PWM and Digital Current Control controllers, and an AC Induction Motor motor:

- > **Field Constant (SLFIELD)**
- > **Slip constant (SLSLIP)**

If the Controller-Drive Interface is Analog ($\pm 10V$) and a Nanomotion Piezoelectric motor:

- > **Dead Zone Minimum (SLDZMIN)**
- > **Dead Zone Maximum (SLDZMAX)**
- > **Zero Velocity Feedforward (SLZFF)**

To start the measurement:

1. Set the parameters to the following initial values
 - > **Low Pass Filter Bandwidth:** 650Hz
 - > **Integrator Gain:** 0
 - > **Velocity Gain:** 100
 - > **Integrator Limit:** 50%
2. Make sure that the **Velocity** field is set to **10**.
3. Click , this enables the motor of the axis you are working on.
4. Click , this starts the motion.
5. Check the Scope display, the ideal display is a square wave as shown in [Figure 9-3](#).

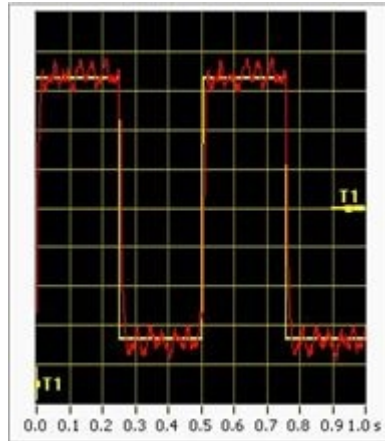


Figure 9-3. Example Scope Display of Well Adjusted Velocity Loop

6. If the jitter is too extreme, adjust the parameters until the **Scope** shows that the motion graph closely matches the reference lines.
 - > Double the **Velocity Gain** until the approximates a square with a small overshoot.
 - > Increase the **Integrator Gain** by hundreds, until you get a narrow overshoot of 10-20% in the waveform.



It is recommended that the **Integrator Bandwidth** not exceed 50Hz. $(\text{Integrator Gain})/20 = \text{Integrator Bandwidth}$

10.2 Position Loop

Once you have adjusted the velocity loop, select **Position Loop** from the **Mode** dropdown list in the Position and Velocity window.



The **Velocity Loop Motion Manager** panel is replaced by the **Position Loop Motion Manager** panel and the Adjuster disables the motor.

The parameters associated with velocity on the Position and Velocity window in the Position Loop mode are:

- > **Position Gain (SLPKP)** - A proportional coefficient of the position.
- > **Position Bandwidth** - Calculated by the **Aduster Wizard**, approximately equal to $\text{SLPKP}/(2 \times \text{PI})$.
- > **Velocity Gain (SLVKP)** - A proportional coefficient that is applied to the velocity
- > **Integrator Gain (SLVKI)** - Integrator coefficient in position servo loop.
- > **Integrator Limit [%] (SLVI)** - Used to prevent the integrator from exceeding a certain value.



The default value, **50%**, is adequate for most systems.

- > **Low Pass Filter Bandwidth (SLVSOF)** - Second order filter bandwidth.
- > **Integrator Bandwidth** - Calculated by Adjuster approximately equal to SLVI/20.5.
- > **Acceleration Feed Forward (SLAFF)** - Used for minimizing positional error during acceleration/deceleration. It can be roughly calculated by $(2^{16} \cdot 1E7 \times EFAC/ACC)/0.1$



You can change the values only of those parameters that are not greyed out.

The **Motion Manager** for Position Loop adjustments serves for controlling the position feedback in order to observe it graphically in the Scope.

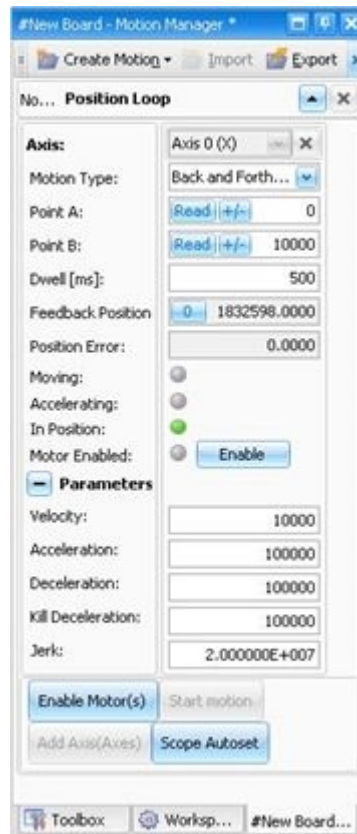


Figure 9-4. Position Loop Manager Panel



The **Motion Type** dropdown list gives you the option of defining the type of motion, the options are:

- > One Direction Repeated Move
- > Back and Forth Move

Position is always measured between two points:

- > **Point A**
- > **Point B**
- > **Point A** is the start position and **Point B** is the end position. The motion is then measured from **Point A** to **Point B**.



You can set these points by manually moving the motor to Point A and clicking , and repeating this for Point B. You can change the sign of the value by clicking .

The **Dwell** value is dwell time between moves.

Feedback Position is the point where the feedback is to be measured. You can reset this position to zero by clicking.

Position Error is a measure of the error in position.

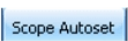
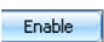
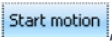
The Motion Parameters that you can manipulate are

- > **Velocity (VEL)**
- > **Acceleration (ACC)**
- > **Deceleration (DEC)**
- > **Kill Deceleration (KDEC)**
- > **Jerk (JERK)**



The values that you put in for these parameters apply only for position loop adjustment using this window. They are separate from the motion profile definitions used by the controller during normal operation.

To start the measurement:

1. Click , this resets the **Scope** to measure the position loop variables: **RVEL**, **FVEL**, and **PE**.
2. Click , this enables the motor of the axis you are working on.
3. Click , this starts the motion.
4. Check the **Scope**, the ideal display is a square wave indicating that the Position Loop is well adjusted.

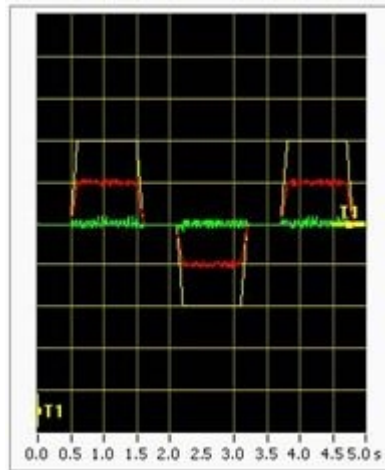


Figure 9-5. Example Scope Display of Well Adjusted Position Loop

5. If the jitter is too extreme, adjust the parameters until the **Scope** shows that the motion graph closely matches the reference lines.
 - > Increase the **Position Gain** to minimize position error during acceleration and deceleration.
 - > Increase the **Acceleration Feedforward** to minimize position error during acceleration

The theoretical Acceleration Feedforward can be calculated as follows.
For a linear motor:

$$SLAFF = 215 \cdot 2 \cdot 10^4 \cdot M / (K_f \cdot K_{res} \cdot I_{peak})$$

Where:

M - Total moving mass [kg]

K_f - Motor force constant [N/A peak]

K_{res} - Encoder resolution [count/revolution]

I_{peak} - Amplifier peak current [A sine peak]



For a rotary motor:

$$SLAFF = 2\pi \cdot 2^{15} \cdot 2 \cdot 10^7 \cdot J / (K_t \cdot K_{res} \cdot I_{peak})$$

Where:

J - Total moving inertia [kgm²]

K_t - Motor torque constant [Nm/A peak]

K_{res} - Encoder resolution [count/revolution]

I_{peak} - Amplifier peak current [A sine peak]

11. Save to Flash

The controller RAM is erased when the controller undergoes a hardware reset. A hardware reset can be user initiated (**HWRES** command through the **Communication Terminal**) 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. The Save to Flash step, where it appears, enables you to save all the data generated by the particular tool to the controller's flash.



If you do not save the data to the controller's flash, all of the data will be lost.

When you enter this step, the Save to Flash prompt is displayed.

Save options			
<input type="checkbox"/> ACSPL+	<input checked="" type="checkbox"/> Configuration	<input type="checkbox"/> SP Programs	<input type="checkbox"/> User Arrays
<input type="checkbox"/> Buffer 0	<input type="checkbox"/> System	<input type="checkbox"/> SP#0	<input type="checkbox"/> I
<input type="checkbox"/> Buffer 1	<input checked="" type="checkbox"/> Axis 0(X)	<input type="checkbox"/> SP#1	<input type="checkbox"/> V
<input type="checkbox"/> Buffer 2	<input type="checkbox"/> Axis 1(Y)	<input type="checkbox"/> SP#2	
<input type="checkbox"/> Buffer 3	<input type="checkbox"/> Axis 2(Z)	<input type="checkbox"/> SP#3	
<input type="checkbox"/> Buffer 4	<input type="checkbox"/> Axis 3(T)		
<input type="checkbox"/> Buffer 5	<input type="checkbox"/> Axis 4(A)		
<input type="checkbox"/> Buffer 6	<input type="checkbox"/> Axis 5(B)		
<input type="checkbox"/> Buffer 7	<input type="checkbox"/> Axis 6(C)		
<input type="checkbox"/> Buffer 8	<input type="checkbox"/> Axis 7(D)		
<input type="checkbox"/> Buffer 9			
> <input type="checkbox"/> D-Buffer			

1. Enter your user name in the **User** field.
2. Enter the application name in the **Application** field.
3. You can, if you desire, enter free text remarks in the **Remarks** field.
4. By default all data is selected. Select the data you want to load into the controller by clicking (thereby deselecting) the checkbox of the data you do not want to be saved.
5. Click **Save**

12. Sin-Cos Encoder Compensation

This chapter covers using the SPiiPlus MMI Application Studio **Sin Cos Encoder Compensation** tool. The tool is used for adjusting SIN-COS encoders that are incorporated in the system.

The **Sin Cos Encoder Compensation** tool enables you to:


- > Select axis for encoder measurements.
- > Display a Sin-Cos Lissajous curve graph of the wave form.
- > Read a controller's gain, phase and offsets compensation parameters (SCGAIN, SCPHASE, SCSOFFS, and SCCOFFS).
- > Write new compensation parameters to the controller.
- > Examine the compensation effect of each parameter.

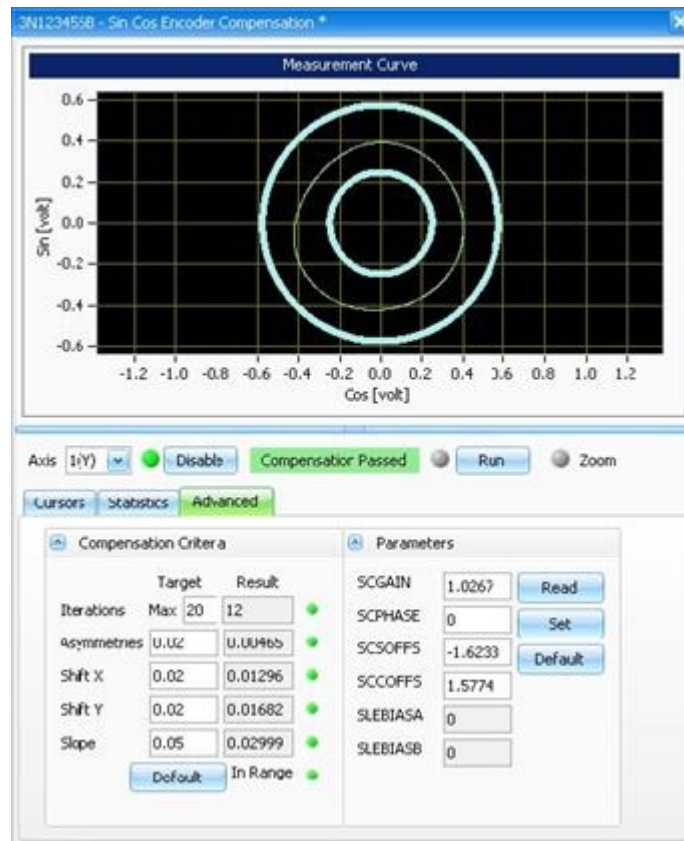
The Sin Cos Encoder Compensation can do the following:

- > Perform step-by-step calculations of the compensation parameters for Sin-Cos encoders based on user input.
- > Perform automatic calculations of the optimum compensation parameters set.

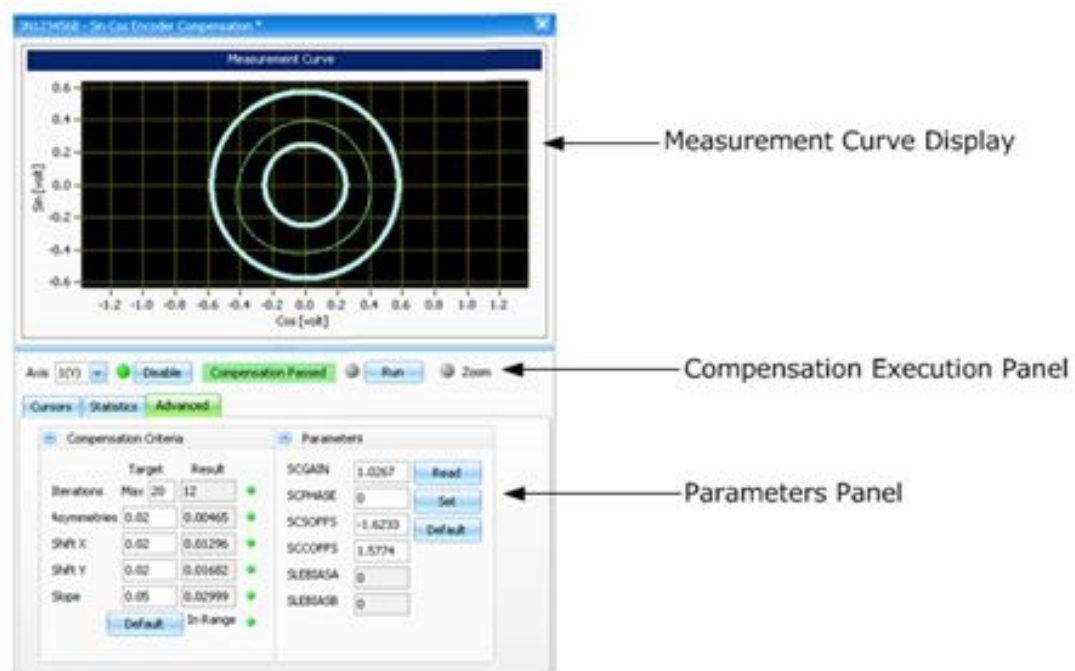
12.1 Activating Sin Cos Encoder Compensation

To activate the **Sin Cos Encoder Compensation** tool:

1. From the Toolbox click **Diagnostics and Monitoring** () to display the Diagnostics and Monitoring tools list.
2. Double-click **Sin Cos Encoder Compensation**. The **Sin Cos Encoder Compensation** window is displayed:



12.2 Sin Cos Encoder Compensation Window



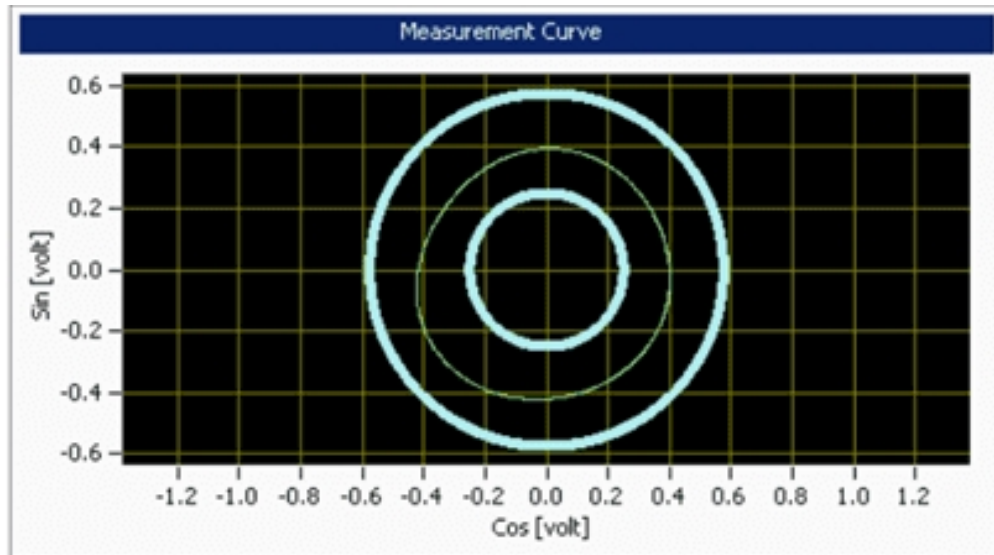
The **Sin Cos Encoder Compensation** window consists of three sections (as shown above):

- > **Measurement Curve Display**

- > Compensation Execution Panel
- > Parameters Panel

12.2.1 Measurement Curve Display

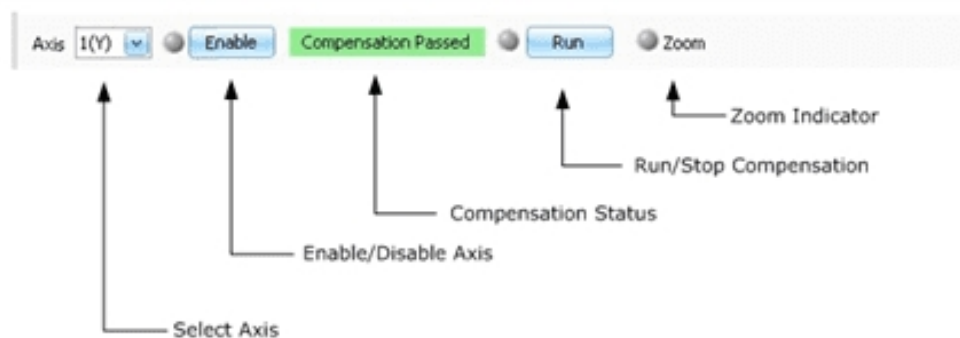
The Measurement Curve is a Lissajous plot, an example of which is shown below, which displays the curve of the results of the measurements performed by **Sin Cos Encoder Compensation**.



The Lissajous plot displays the aspect ratio of the phase shift between the input and output, with an aspect ratio of 1 (perfect circle) corresponding to a phase shift of $\pm 90^\circ$ as well as amplitude differences (ellipse).

12.2.2 Compensation Execution Panel

The Sin Cos Encoder Compensation Execution panel:



serves for executing the **Sin Cos Encoder Compensation** calculation routines. The elements in this panel are:

- > **Select Axis** - used for selecting the axis from a dropdown list, the contents of which depend on the type of controller. There is a LED associated with the selected axis that can be:
 - > Grey - the axis is not enabled

- > Green - the axis is enabled.
If the axis is not enabled, you enable it by clicking the **Enable** button.



If you select an axis that is not connected to a Sin-Cos Encoder, it cannot be enabled.


Select the required axis from the dropdown list, for example:



- > **Compensation Status** - this field displays messages associated with the **Sin Cos Encoder Compensation** status. The messages that can be displayed are:
 - > **Compensation Off** - the Sin Cos Encoder Compensation has not been activated. (You activate it by clicking the **Run** button).
 - > **Compensation in Progress** - the Sin Cos Encoder Compensation is performing its measurements.
 - > **Compensation Passed** - the Sin Cos Encoder Compensation has completed its measurements and the compensation is correct.
 - > **Compensation Failed** - the Sin Cos Encoder Compensation has completed its measurements and the compensation is not correct.
- > **Compensation Cancelled** - the Sin Cos Encoder Compensation has been stopped (when you click the **Stop** button) before the measurements have been completed.
There is a LED associated with the status which can be:
 - > Grey - the Sin Cos Encoder Compensation has not been activated, or has completed its measurements.
 - > Green/blinking - the Sin Cos Encoder Compensation has been activated and is working.
- > **Run/Stop** button - activates or halts the Sin Cos Encoder Compensation measurements, when clicked, the **Run** button changes to the **Stop** button for manually halting the measurements in progress.
- > **ZoomIndicator** - you have the following Zoom options:



When the display has been zoomed, the Zoom LED (●) turns green.

- > Zoom In - Hold Shift key and place the mouse cursor in the Scope display in area you want to zoom into. Click the Left mouse button to display the Zoom icon. When the icon appears, click the Left mouse button. Without releasing the Shift key, each time you click the Left mouse button, you increase the zoomed display.
- > Zoom In Area - While holding the Shift key, press the Left mouse button and drag to mark the area you want to zoom, then release the mouse button.
- > Zoom Out - While holding the Shift key, place the mouse cursor in the zoomed area and click the Right mouse button. Without releasing the Shift key, each time you click the Right mouse button, you zoom out one level.
- > Move Graph - Hold Ctrl key and Left mouse button - a hand is displayed. Drag the graph to where you want.
- > Undo - To undo all zooms, press Shift + Backspace keys. (You can also undo the zoom by clicking the Zoom LED .

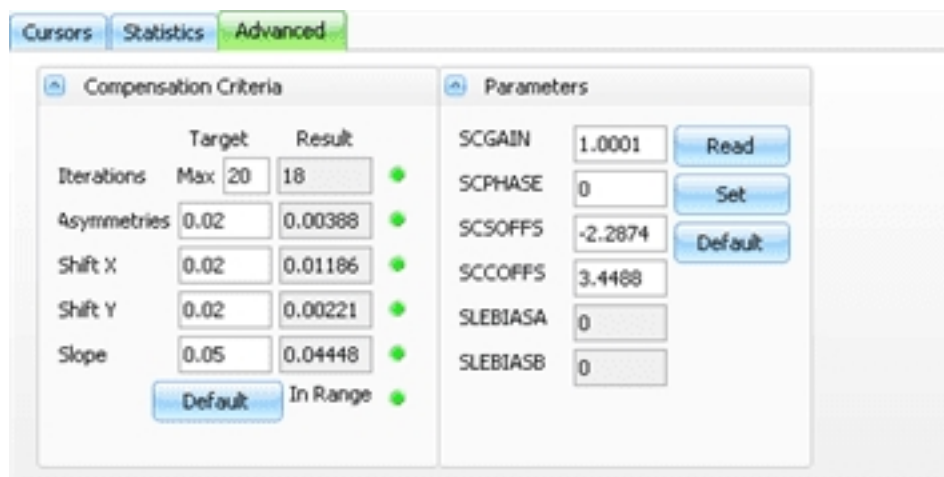
12.2.3 Parameters Panel




You use the Parameter panel to set up parameters and display options you want. The panel has three tabs:


- > **Advanced**
- > **Cursors**
- > **Statistics**

12.2.3.1 Advanced Tab

The Measurement tab is used for defining the required input measurement parameters and to control the output results.



Compensation Criteria			
	Target	Result	
Iterations	Max 20	18	
Asymmetries	0.02	0.00388	
Shift X	0.02	0.01186	
Shift Y	0.02	0.00221	
Slope	0.05	0.04448	
Default In Range			

Parameters	
SCGAIN	1.0001 
SCPHASE	0 
SCSOFFS	-2.2874 
SCCOFFS	3.4468
SLEBIASA	0
SLEBIASB	0

The panel is divided into two sections:

- > Compensation Criteria

This section displays the target criteria used by the **Sin Cos Encoder Compensation** when making the calculations and current values of calculated parameters along with a set of LEDs. If the current value of any parameter is equal to or less than the target value, the

associated LED is Green (otherwise it is Red if the value fails, or Grey if the parameter is not relevant).

The criteria are:

- > Iterations - the number of iterations that the **Sin Cos Encoder Compensation** is to perform during the measurements.
- > Asymmetries - $\text{Max}(dx, dy) - \text{Min}(dx, dy)$, where dx is $X_{\text{final point}} - X_{\text{start point}}$, and dy is $Y_{\text{final point}} - Y_{\text{start point}}$. If this is zero, the graph is a perfect circle.
- > Shift X - the offset on the X axis
- > Shift Y - the offset on the Y axis
- > Slope - dy/dx , where dx is $X_{\text{final point}} - X_{\text{start point}}$, and dy is $Y_{\text{final point}} - Y_{\text{start point}}$.

You can make changes to the values of the **Target** fields when fine-tuning the compensation.



Clicking **Default** sets default values of the target calculated parameters.

> Parameters

This section lists the ACSPL+ variables that are connected with managing Sin-Cos encoders. These are:

- > **SCGAIN** - Modify the amplitude of the COS signal, to compensate amplitude difference between the SIN/COS signals
- > **SCPHASE** - Modify the phase of the COS signal, to compensate phase difference between the SIN/COS signals (phase should be 90 degrees)
- > **SCSOFFS** - Compensating SIN signal offset
- > **SCCOFFS** - Compensating COS signal offset
- > **SLEBIASA** - Defines a Sin-Cos encoder's hardware compensation for the Sine offset
- > **SLEBIASB** - Defines a Sin-Cos encoder's hardware compensation for the Cos

SCGAIN, **SCPHASE**, **SCSOFFS**, and **SCCOFFS** are all software-based variables.

The **SLEBIASA** and **SLEBIASB** variables, on the other hand, are hardware-based. Hardware compensation has some advantages over software compensation: possibility to get analog signals out of saturation, and making hardware based features like PEG more accurate.

In principle these variables have similar functionality as **SCSOFFS** and **SCCOFFS**; however, they are only supported by the following controller models:



- > SPiiPlusNT-HP/LD (compensation diapason ± 350 mV)
- > SPiiPlusDC-HP/LD (compensation diapason ± 350 mV)
- > SPiiPlus-UDMpc (compensation diapason ± 625 mV)
- > PiiPlus-CMnt-2 (compensation diapason ± 750 mV)

If the controller model supports it, **Sin Cos Encoder Compensation** first calculates the software compensation variables, and then writes the final values to the hardware variables (and resets the software ones).

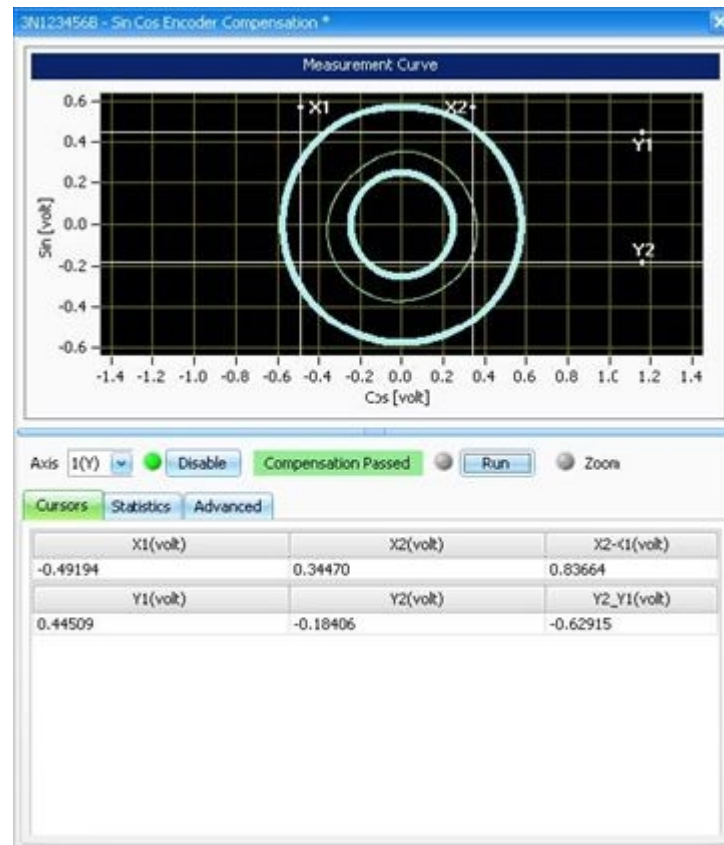
You have the option of changing the values by inputting new values in the appropriate fields. Details of these variables are provided in the *SPiiPlus Command & Variable Reference Guide*.

There are three action buttons associated with this section:

	Sin Cos Encoder Compensation reads the current variable values from the controller.
	Sin Cos Encoder Compensation writes the variable values into the controller.
	Sin Cos Encoder Compensation sets the default values into the controller.

12.2.3.2 Cursors Tab

The **Cursors** tab enables you to mark points on the display where you are interested in seeing exactly what the values at the points are.



There are two cursors, **X1** and **X2**, for delimiting the Cos voltage values, and two cursors, **Y1** and **Y2**, for delimiting the Sin voltage. As the **Sin Cos Encoder Compensation** performs its routines, the values at the cursor settings are displayed in the field below the graph. For setting the cursors see *SPiiPlus MMI Application Studio User Guide*.

12.2.3.3 Statistics Tab

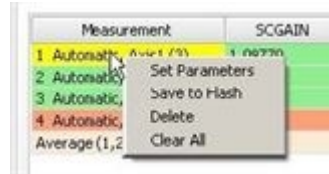
For each run, the **Sin Cos Encoder Compensation** builds a statistics table consisting of compensation calculation measurements in the Statistics tab:

Cursors Statistics Advanced							
Measurement	SCGAIN	SCPHASE	SCSOFFS	SCCOFFS	SLEBIASA	SLEBIASB	Start Position
1 Axis 1, Auto Com...	1.26170	2.00000	4.00000	8.67510	0.00000	0.00000	6643526224...
2 Axis 1, Auto Com...	1.07580	-1.97600	-1.13170	5.51590	0.00000	0.00000	6643526224...
3 Axis 1, Auto Com...	1.05370	3.00000	-0.19910	4.85860	0.00000	0.00000	6643526224...
4 Axis 1, Auto Com...	1.02670	3.00000	-1.62330	1.57740	0.00000	0.00000	6643526224...
5 Axis 1, Auto Com...	1.00010	3.00000	-2.28740	3.44880	0.00000	0.00000	6643526224...
6 Axis 1, Auto Com...	1.00010	3.00000	-2.28740	3.44880	0.00000	0.00000	6643526224...
Average, Axis1 (2,3...	1.03128	-0.39520	-1.50578	3.76990	0.00000	0.00000	

The rows are color-coded, Green for those values that passed the compensation measurements, and Red for those that failed.

At the bottom of the table the **Sin Cos Encoder Compensation** adds a line containing the average values of the variables. There is a separate line for each axis that was checked.

Right-clicking on any row displays a menu with the following options:



- > **Set Parameters** - selecting this option causes the **Sin Cos Encoder Compensation** to use these variable values for purposes of making the measurement.
- > **Save to Flash** - selecting this option causes **Sin Cos Encoder Compensation** to save the values to the controller's flash memory.
- > **Delete** - selecting this option deletes the row from the Statistics.
- > **Clear All** - selecting this option removes all the values from the Statistics.

12.2.4 Running Sin Cos Encoder Compensation

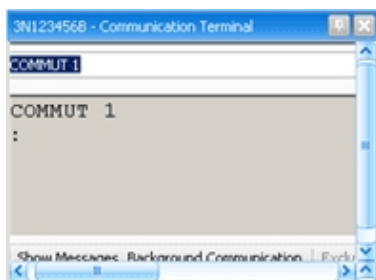
To perform automatic encoder compensation:

1. From the Execution pane select axis from the **Axis** dropdown list.
2. Click **Enable**
3. Click **Run**

If the motor associated with the encoder is a brushless motor, it has to be commutated before encoder compensation. In the event that it has not been commutated, the following is displayed:



The easiest way to commutate the motor at this point is to enter the **COMMUT** command via the **Communication Terminal**, for example:



Then click **Run** again.

4. If the compensation is successful, go to the **Advanced Tab** of the Parameter Selection panel and save the values to the controller's flash memory by clicking **Set** in **Parameters**.



Once you have completed the Sin-Cos compensation, return to the [Verification](#) Task of the **Adjuster Wizard**.

13. 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).

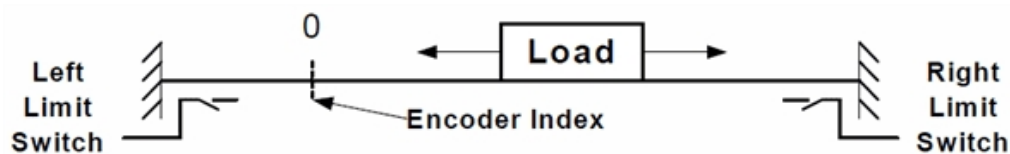


Figure 12-1. Homing



1. If the axis is equipped with a DC Brushless motor, then the homing program should also include a commutation startup program, see [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.



Version 2.60 has a set of predefined homing methods. You can also define a homing method as well. The homing methods obey the DS402 standard.
For details, see *SPIIPlus ADK Suite v2.60 Release Notes*.

13.1 Homing Program Using Limit Switches

The following program is an example of a homing procedure for the 0 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 (in this case: 0).
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

```

13.2 Homing Program Using Hard Stops

The following program is an example for a homing procedure of the 0 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 (in this case: 0).
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

```

14. Protecting Application

Up to this point, the controller has been in the 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.


The controller can be in one of two modes:

- > Protected Mode - certain restrictions are placed on the programs in the controller's Program buffers. Usually this mode is used to ensure that the programs cannot be edited.
- > Not Protected Mode - no restrictions are placed on the Program buffers.

The **Protection Wizard** is used for setting and unsetting the Protection Mode of the controller. It leads you through the performance of the following tasks:

- > Define Protection
- > Update Protection
- > Remove Protection
- > View Protection

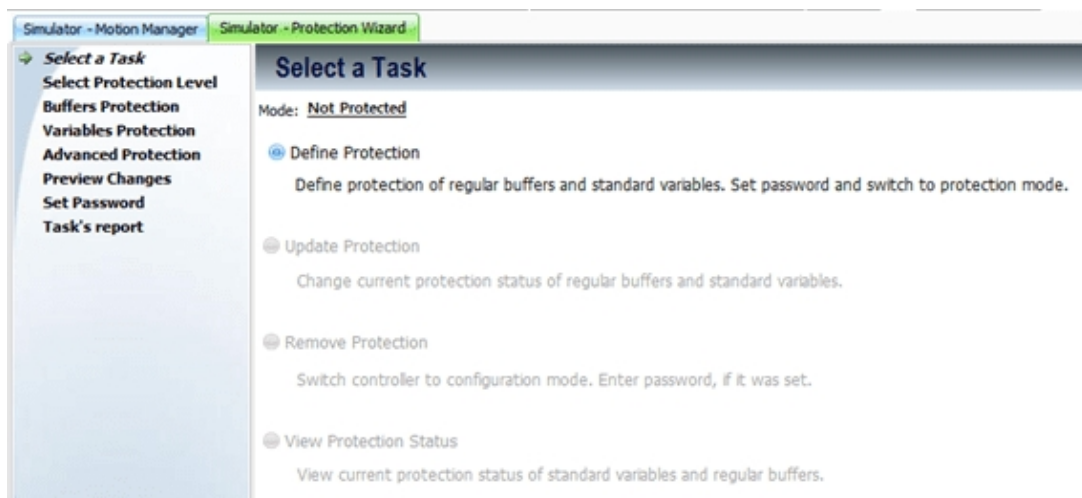
14.1 Starting Protection Wizard

1. From the Toolbox click **Application Development** () to display the Application Development tools.
2. Click **Protection Wizard** in the Application Development list of tools.



You can also activate the **Protection Wizard** using the right-click **Add Component** option of the Workspace Tree.

The **Protection Wizard** Task window is displayed in the workspace.



3. Select the task you want to perform by clicking the appropriate button and then click **Next**.



If the controller is in the **Not Protected** mode, the only task available is **Define Protection**.

If the controller is in the **Protected** mode, the tasks available are: **Update Protection**, **Remove Protection**, and **View Protection Status**.

14.2 Define Protection

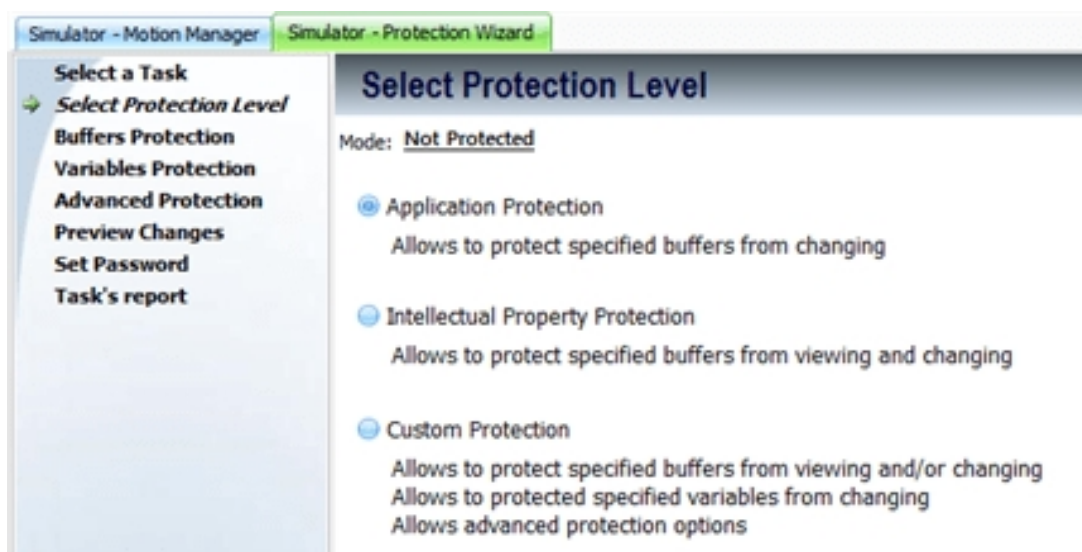
The Define Protection task enables you to put the controller into the Protected Mode and to set certain restrictions on the programs stored in the controller's buffers.

To define Protection:



This task, and this task only, is available if the controller is in the **Not Protected** mode.

1. Since this is the only task available, click **Next**. The Set Protection Level window is displayed.



There are three tasks that can be performed from the Select Protection Level window:

- > Application Protection
You perform this task to set the program in the specified buffers to Read Only, that is, it can be viewed but cannot be modified.
- > Intellectual Property Protection
You perform this task to set the program in the specified buffers such that its contents cannot be modified and cannot be seen.
- > Custom Protection
There are three tasks that can be performed from the Custom Protection window:

- > You can set the program in the specified buffers such that its contents cannot be modified and/or cannot be seen.
- > You can set specified variables to Read Only, that is, it can be viewed but cannot be modified.
- > You allow the system to be reconfigured.

14.2.1 Setting Application Protection

1. To set Application Protection, select the button next to **Application Protection**. Then click **Next**. The Buffers Protection task is displayed.



By selecting the **Prevent from editing** checkbox you set the program in the buffer to Read Only, that is, it can be viewed but cannot be modified.

2. Click **Next**. The Preview Changes window is displayed:



3. Check that the proper Protections are set. If they are, select **Accept**. Click **Next**. The Set Password window is displayed.



4. If you want to set a user password to prevent unauthorized personnel from removing Protection, type it into the **Enter password** and **Confirm password** fields.



A password is not required.

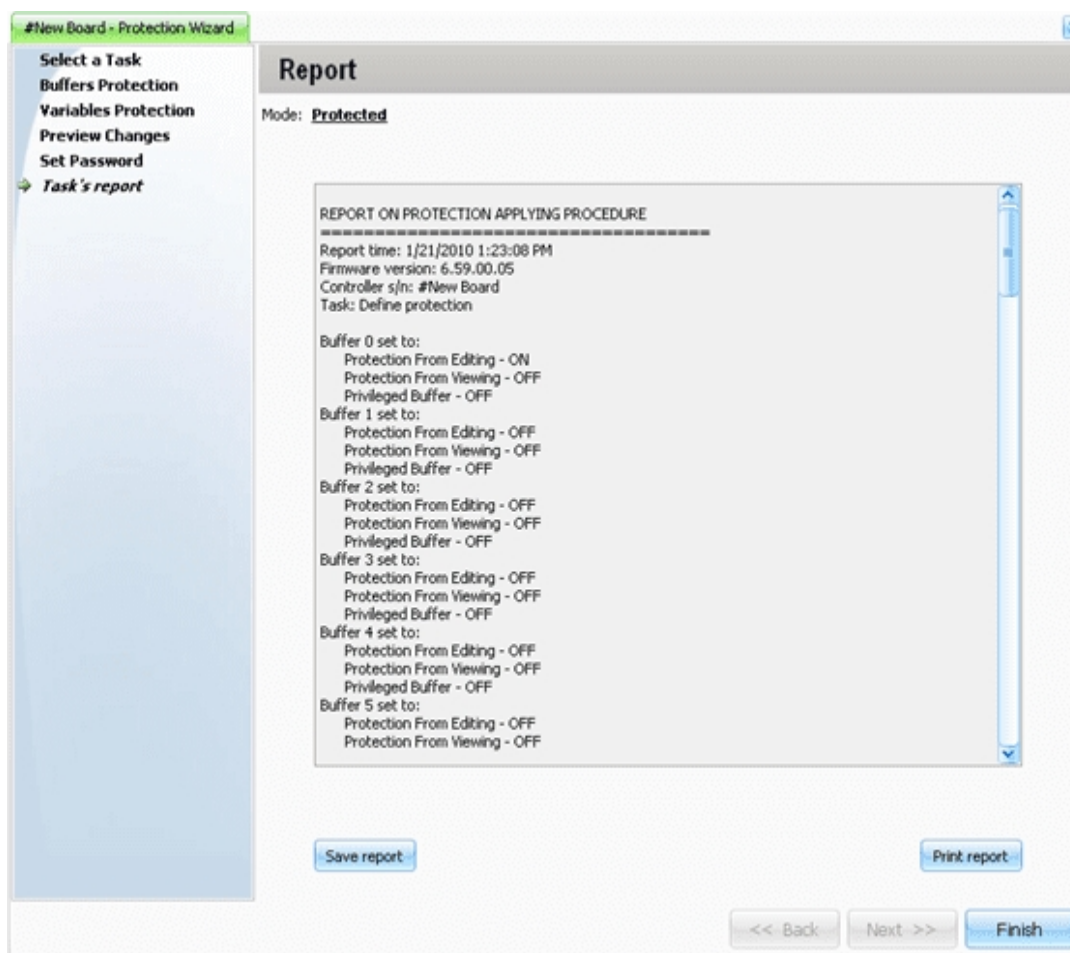
5. Click **Next**. The controller is rebooted. Wait for the notification of a successful restart.



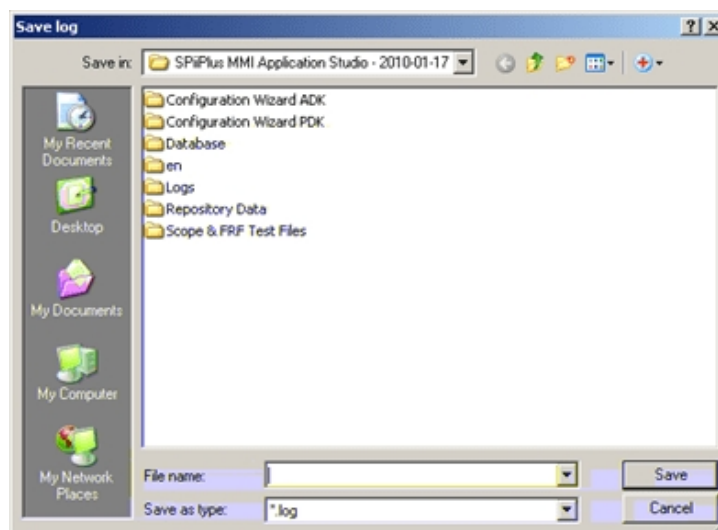
The controller must be restarted, otherwise the protected programs can become corrupted.

6. Click **OK**.

The Report window is displayed.



7. To save the report click **Save report**. The browser window is displayed:



8. Select a directory and enter a filename. Click **Save**.

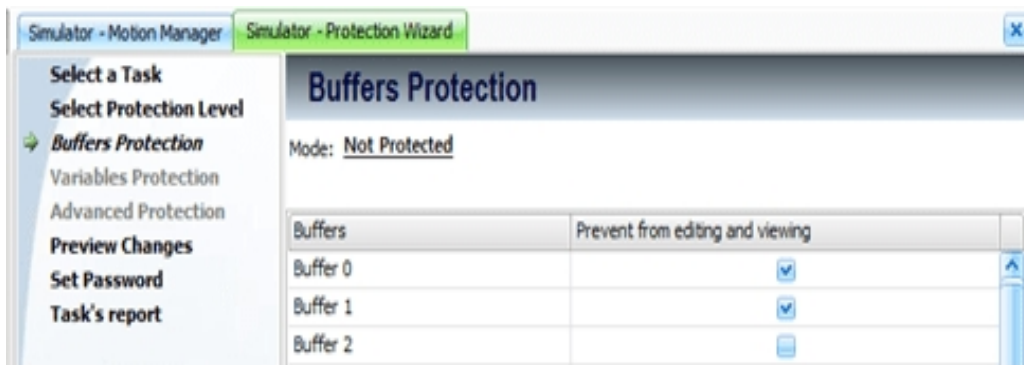


The file extension for all reports is **.log**.

9. To print the report click **Print report**.
10. Click **Finish** to exit the Protection Wizard.

14.2.2 Setting Intellectual Property Protection

1. To set Intellectual Property Protection, select the button next to **Intellectual Property Protection**. Then click **Next**. The Buffers Protection task is displayed.



By selecting the **Prevent from editing and viewing** checkbox, you set the program in the specified buffers such that its contents cannot be modified and cannot be seen.

2. Click **Next** to preview changes and follow step 3 through step 10 in [Setting Application Protection](#) to complete setting protection and to exit the Protection Wizard.

14.2.3 Setting Custom Protection

14.2.3.1 Custom buffer protection

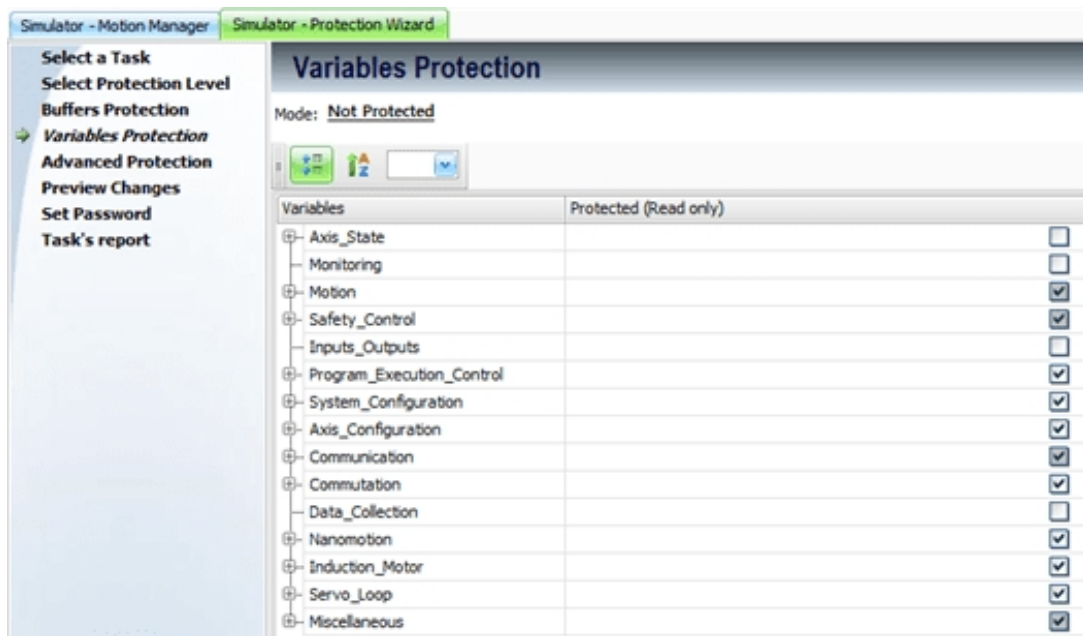
1. To set Custom Protection, select the button next to **Custom Protection**. Then click **Next**. The Buffers Protection task is displayed.



By selecting the **Custom Protection** checkbox, you set the program in the specified buffers such that its contents cannot be modified and/or cannot be seen.


2. Click **Next**. The Variables Protection Task is displayed.

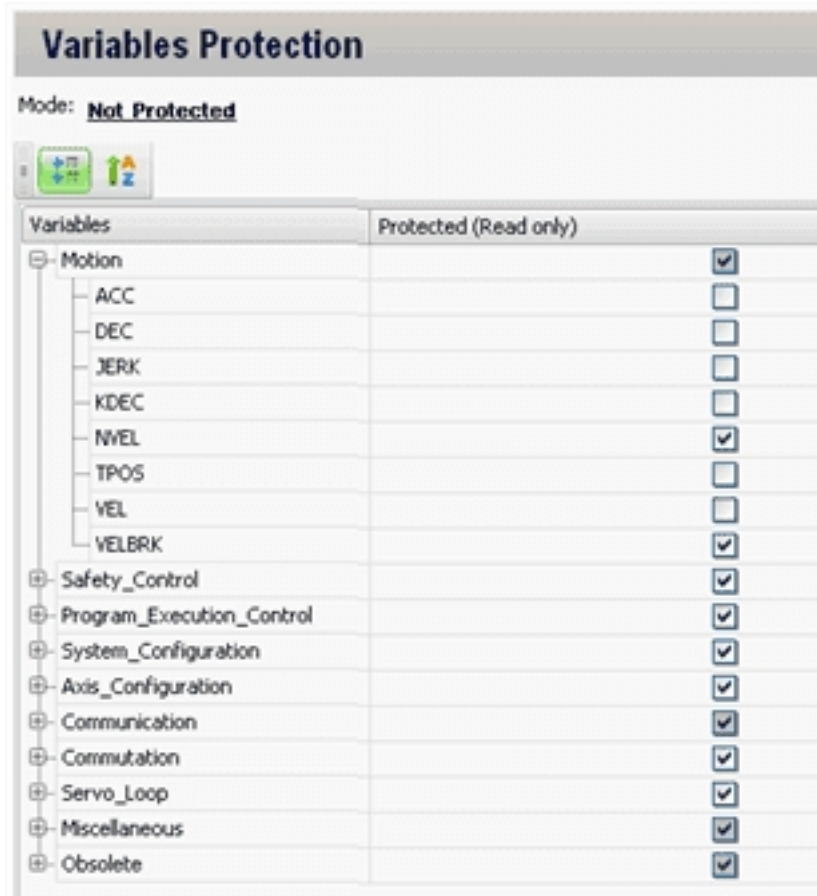
14.2.3.2 Custom variable protection



You can click  to sort the variables by groups, or  to sort alphabetically.

1. Select the appropriate **Read only** box to prevent the variables from being edited.

You can either set all the variables in a group by selecting the **Read only** box, or expand the groups by clicking the tree expansion button:  and marking individual variables for read only.



2. Click **Next**. The Advanced Protection Task is displayed.

14.2.3.3 Advanced protection options



1. To allow system configuration, select the checkbox.
2. Click **Next** to preview changes and follow step 3 through step 10 in [Setting Application Protection](#) to complete setting protection and to exit the Protection Wizard.

14.2.4 Update Protection

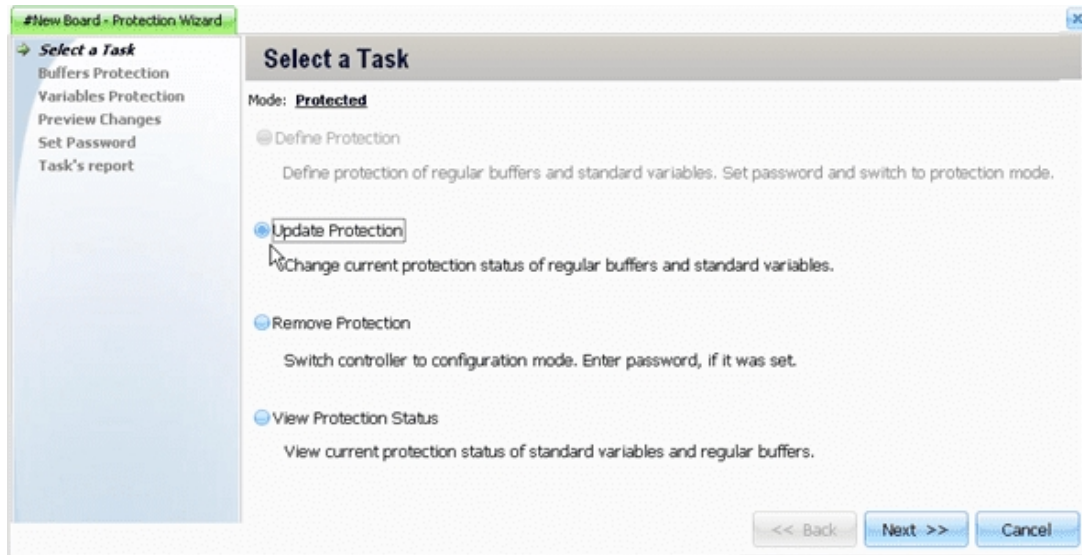
The Update Protection task enables you to make changes in Protection settings.



This task is available only if the controller is in the **Protected** mode.

To update Protection data:

1. Select **Update Protection** in the Select a Task window.




2. Click **Next**. The Set Protection Level window is displayed.



3. You have the option of changing the Protection by selecting the appropriate buttons:
 - > To change Application Protection settings, select the button next to **Application Protection**. Then click **Next** and follow the procedure described in [Setting Application Protection](#).

- > To change Intellectual Property Protection settings, select the button next to **Intellectual Property Protection**. Then click **Next** and follow the procedure described in [Setting Intellectual Property Protection](#).
- > To change Custom Protection settings, select the button next to **Custom Protection**. Then click **Next** and follow the procedure described in [Setting Custom Protection](#).



In the selected protection task, you enable protection or cancel an existing Protection by clicking the selected checkbox () to toggle it on or off.

14.2.5 Remove Protection

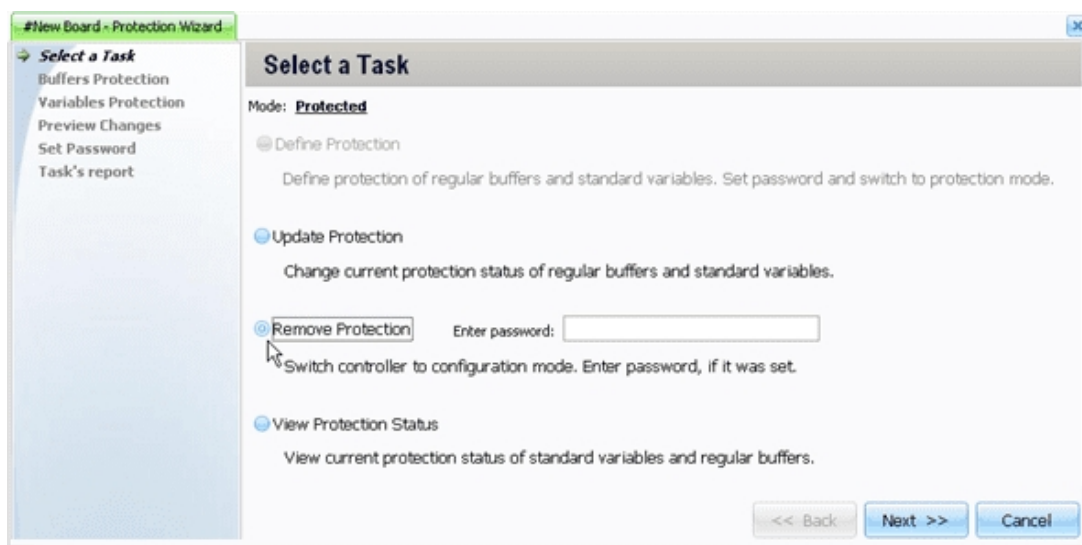
The Remove Protection task enables you to disable the **Protection** mode.



This task is available only if the controller is in the **Protected** mode.

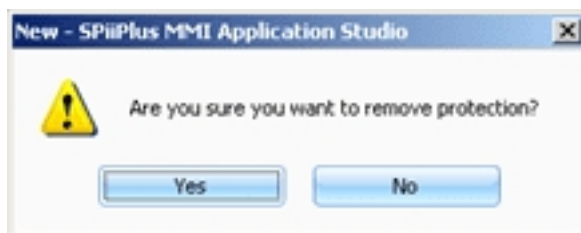
To remove Protection:

1. Select **Remove Projection** in the Select a Task window.

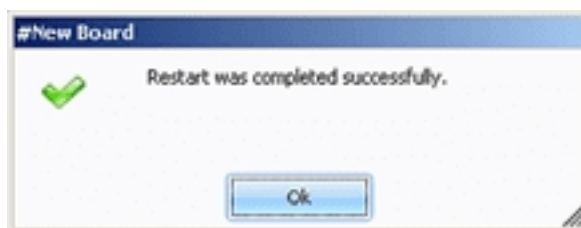


The password field appears next to the task heading when you select this task.

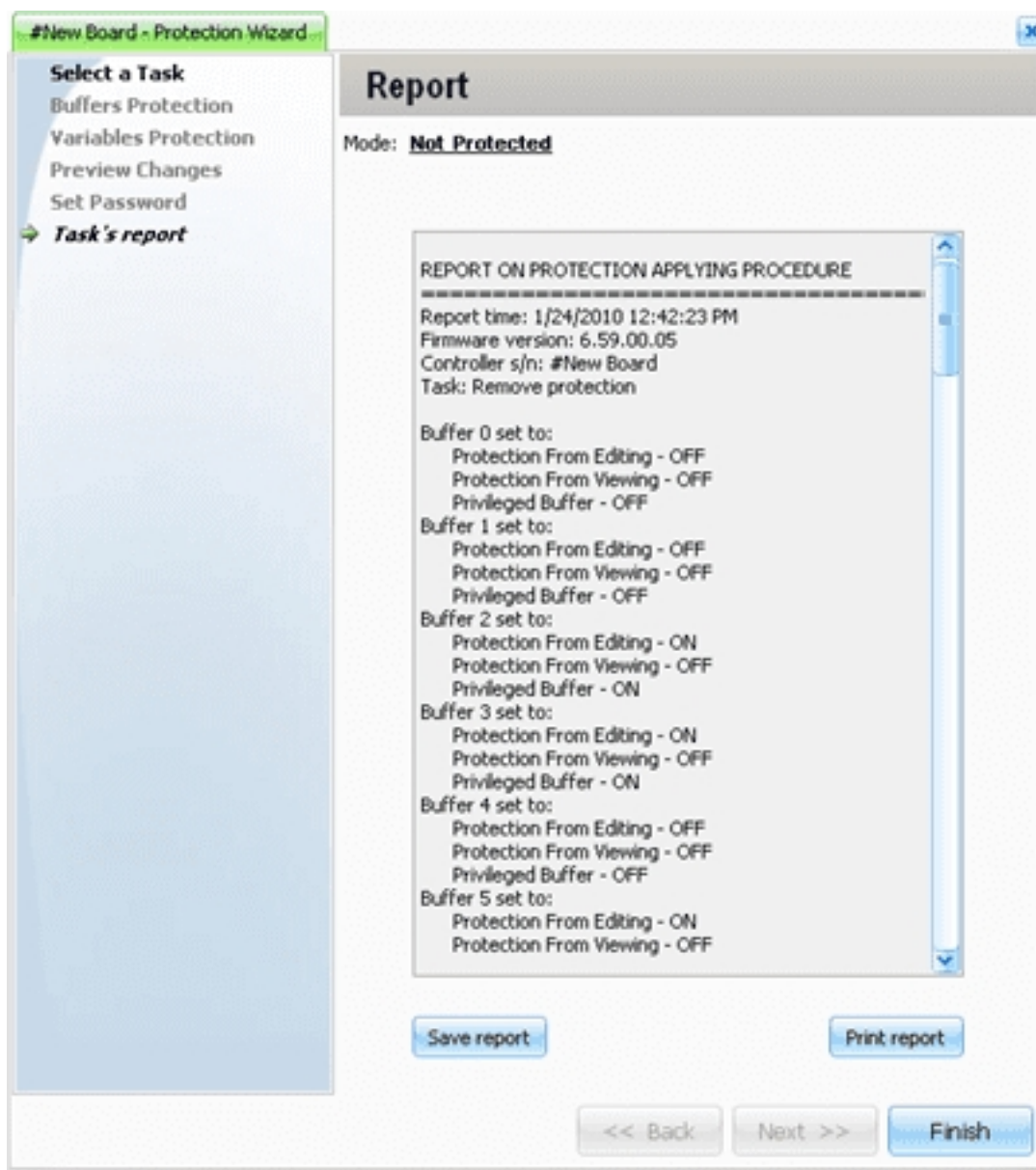
2. Type in the password, if needed, and click **Next**. The following notification appears:



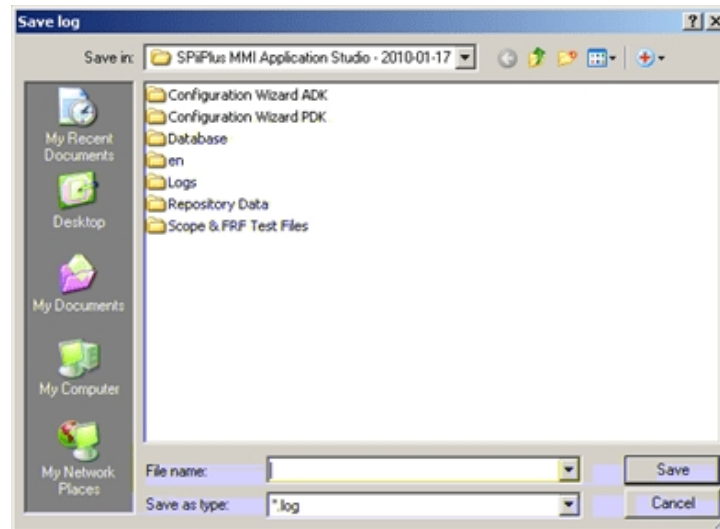
3. Click **Yes**. The controller reboots.
4. Wait for the appearance of a successful restart message:



5. Click **OK**. The Report window is displayed:



6. To save the report click **Save report**. The browser window is displayed:



Select a directory and enter a filename. Click **Save**.



The file extension for all reports is **.log**.

7. To print the report click **Print log**.
8. Click **Finish** to exit the Protection Wizard.

14.2.6 View Protection

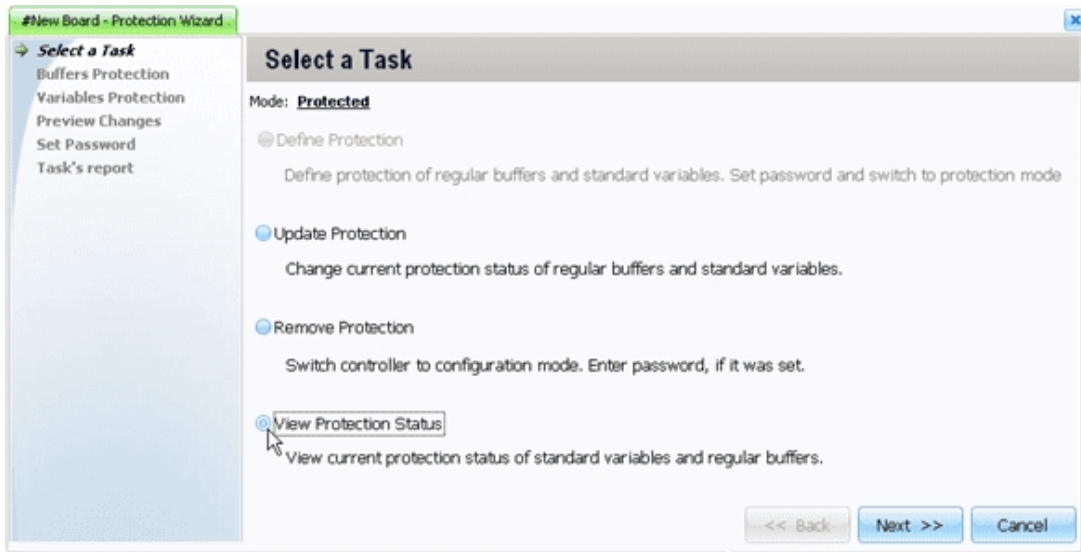
The View Protection task enables you to view the Protection mode settings of the controller.



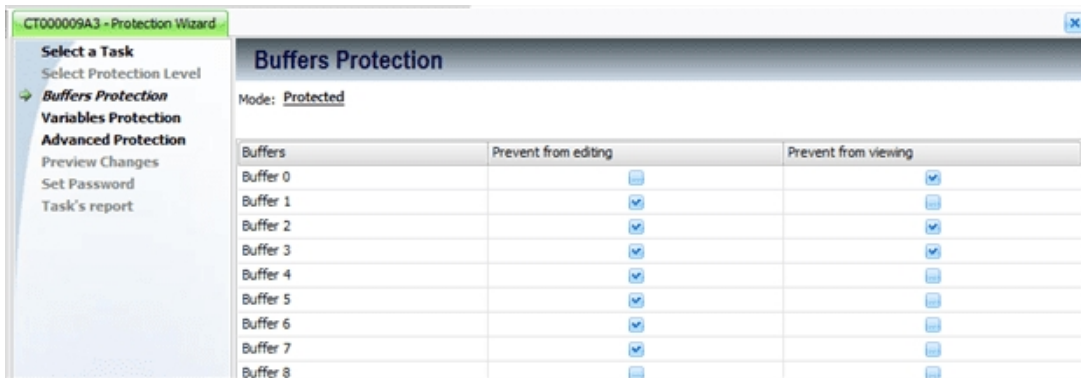
This task is available only if the controller is in the **Protected** mode.

To view the Protection mode settings:

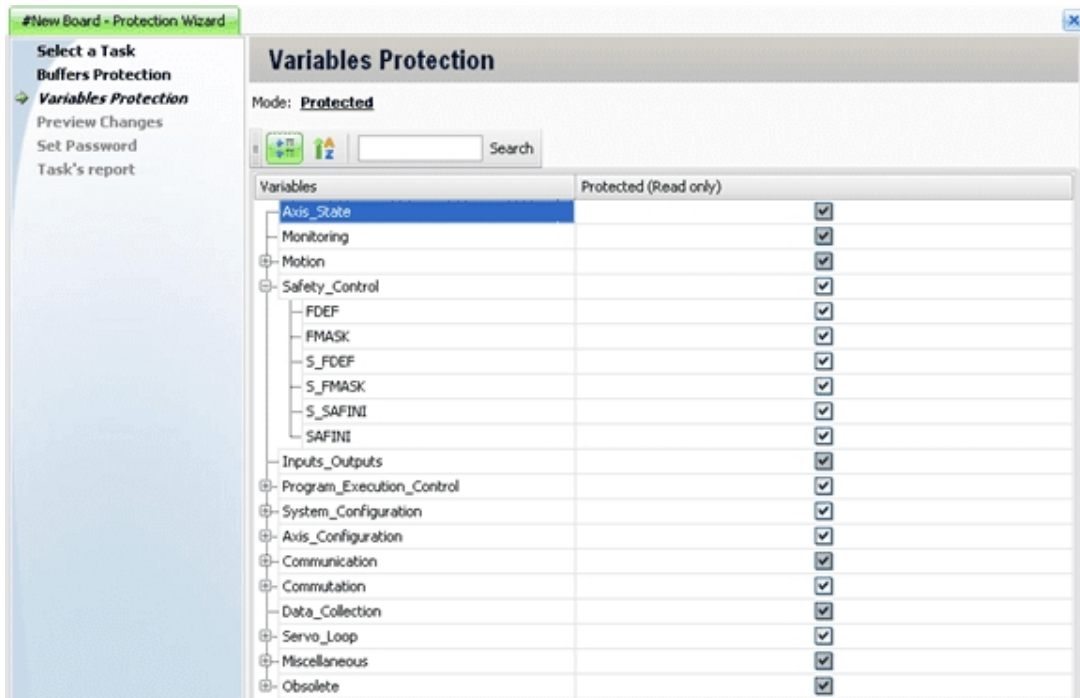
1. Select **View Projection** in the Select a Task window.



2. Click **Next**. The Buffers Protection task is displayed.

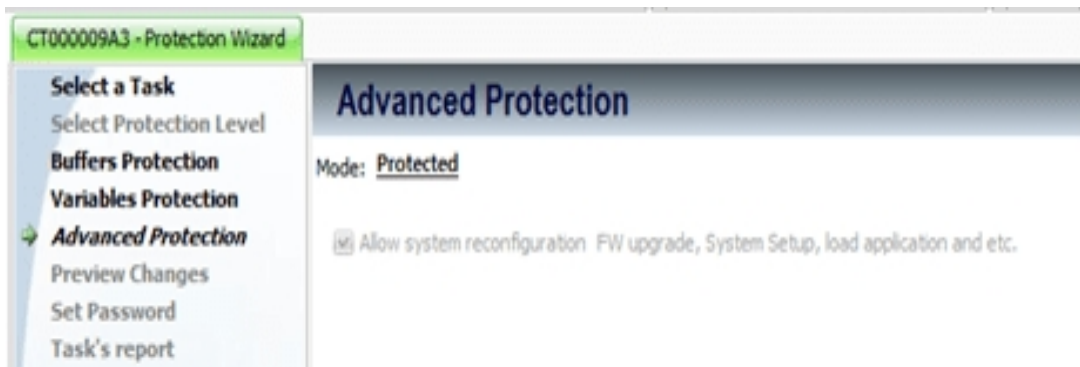


3. Click **Next**. The Variables Protection is displayed.



You can view the settings of the individual variables by clicking the tree expansion button: of the groups.

- Click **Next**. The Advanced Protection is displayed.



- Click **Finish** to exit the Protection Wizard.

14.2.7 Entering Incorrect Password

If the password that you enter is incorrect, the **Protection Wizard** displays the following message:



1. Acknowledge the message by clicking **Continue**. The wizard informs you of the Protection task that has failed, for example:



2. Your options are:
 - > Click **Back** to return to the window with the password field and enter the correct password, or
 - > Click **Finish** to exit the wizard.

Appendix A. Working with a Gantry Axis

A.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 14-1 illustrates a typical gantry system with two motors (X1, X2) that move a single axis (X):

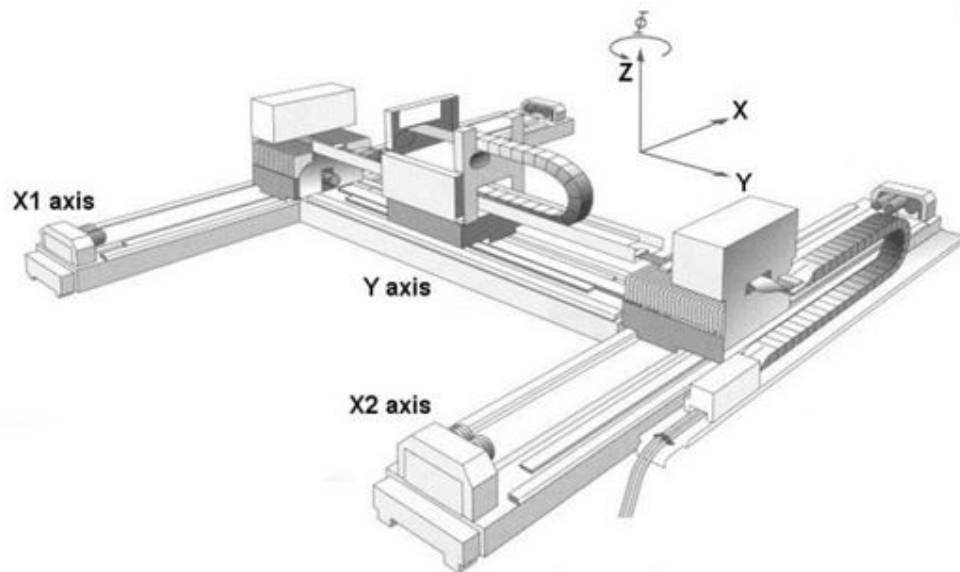


Figure 14-1. Typical Gantry

A.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 Figure 14-2). 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:

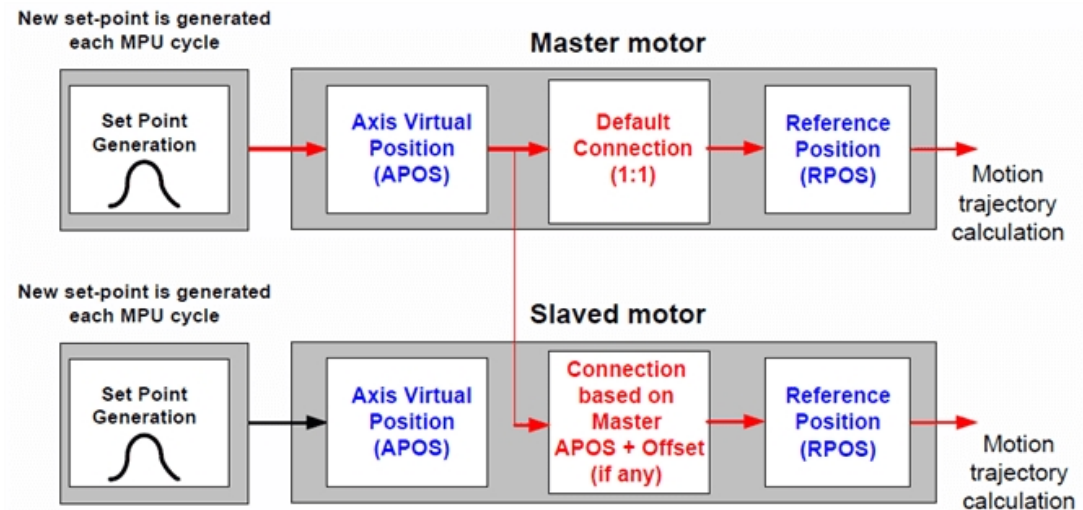


Figure 14-2. Gantry Axis Profile Generation

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 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.

A.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.

A.3.1 Gantry Setup and Configuration

1. **SPiiPlus MMI Application Studio > Toolbox > Setup > Adjuster:** Set the safety parameters for the master (limits, current values, position error etc). Copy the configuration to the identical slaved axis – see [Axis Architecture](#).
2. **SPiiPlus MMI Application Studio > Toolbox > Setup > Adjuster > Commutation:** make commutation separately for each (if DC brushless motor is being used) - see [Commutation](#).
3. **SPiiPlus MMI Application Studio > Toolbox . Setup > Adjuster > Open loop verification:** make verification in open loop separately for each motor - see [Open Loop and Position Verification](#).
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)
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 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 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.

A.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

```

A.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)

Appendix B. Dual Loop Control

Dual Loop control is commonly used when there is poor stiffness between the motor and the load (for example, in belt-driven systems as shown in [Figure 15-1](#)). 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.

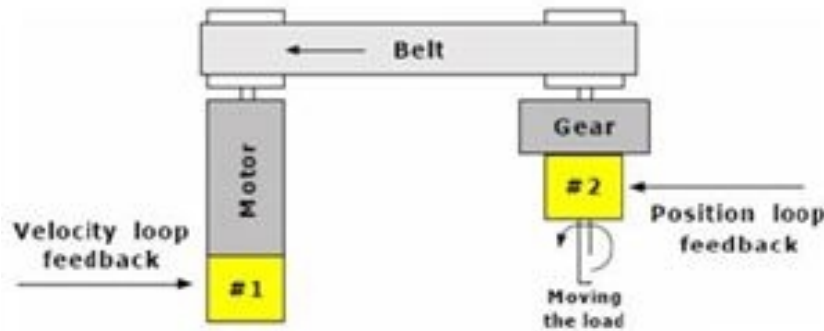


Figure 15-1. Dual Loop System



The Dual Loop tuning process described in this chapter is compatible with NT Firmware version 2.1 or higher.

Normally the SPiiPlus MMI Application Studio **Adjuster Wizard** is used for fine-tuning control loops; however, the current version does not support Dual Loop control. This chapter provides a work-around procedure for setting up Dual Loop control.

B.1 Dual Loop Basics

Dual Loop control of an axis in NT products is more flexible compared to previous ACS Motion Control products. The user can assign feedbacks from various channels or from an analog input to the axis, with the constraint that the axis and channels must relate to the same Servo Processor.

In Dual Loop the control of the axis is based on at least two feedback sources:

- > Load position feedback - used as input for the Position Loop.
- > Motor position feedback - used as input for the Velocity Loop and for the motor's Commutation as well.

[Figure 15-2](#) a simplified block diagram of Dual Loop control.

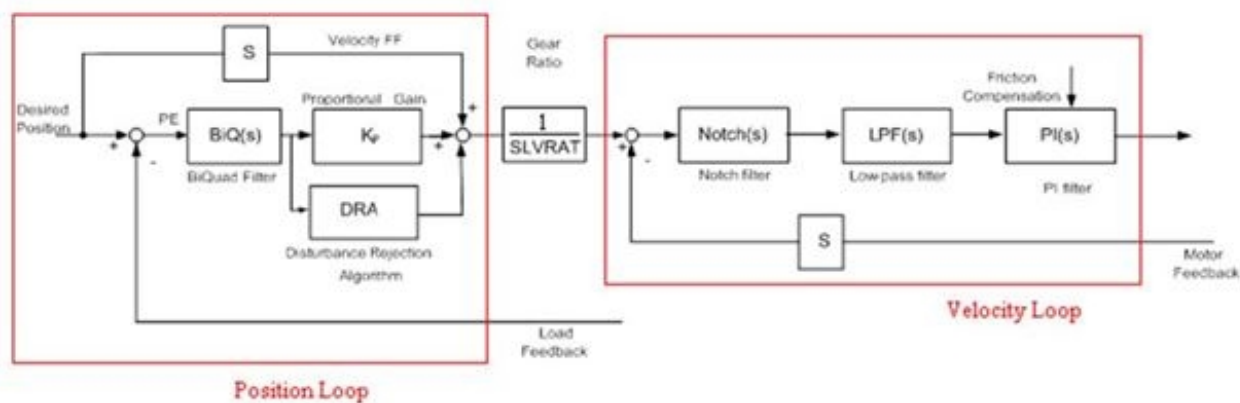


Figure 15-2. Controller in Dual Loop Block Diagram

B.2 ACSPL+ Variables

The following ACSPL+ variables are involved in Dual Loop control:

B.2.1 MFLAGS

MFLAGS is an integer array, with one element for each axis in the system, each element of which contains a set of bits used for configuring the motor.

In order to set the system for Dual Loop control, bit 20 (#DUALLOOP) of **MFLAGS** has to be set to 1 for the relevant axis, for example, $MFLAGS(0).\#DUALLOOP = 1$, sets axes 0 to Dual Loop control, that is, the Bi-Quad filter is moved from the Velocity Loop to the Position Loop.

B.2.2 SLVRAT

SLVRAT is a real array, with one element for each axis in the system, and is used for defining the gear ratio between the Position Loop feedback and the Velocity Loop feedback:

$$SLVRAT = \frac{\text{position resolution}}{\text{velocity resolution}}$$

B.2.3 SLVKP

SLVKP is a real array, with one element for each axis in the system, and is used for providing a proportional coefficient that is applied to the velocity of the specified axis.

B.2.4 XVEL

XVEL is a real array, with one element for each axis in the system, and is used for defining the maximum allowed velocity for the axis. **XVEL** affects the effective value of the Velocity Loop proportional gain **SLVKP** differently:

In Dual Loop, the effective value is given by:

$$SLVKP_{\text{effective}} = SLVKP \cdot \frac{2^{21}}{XVEL \left[\frac{\text{motor feedback}}{\text{sec}} \right]} = SLVKP \cdot \frac{2^{21}}{XVEL \left[\frac{\text{load feedback}}{\text{sec}} \right]} \cdot SLVRAT$$

If $XVEL \left[\frac{\text{load feedback}}{\text{sec}} \right] > 2^{21}$ then the effective gain is decreased according to the latter formula. Otherwise the effective gain is increased.

B.2.5 EFAC

EFAC is a real array, with one element for each axis in the system, and is used for defining a factor between the raw feedback in encoder counts and the **FPOS** value calculated by the controller.

Its format is:

EFAC(axis_index) = value

Where value ranges between 1^{-15} to 1^{+15} (the default value is 1).

B.2.6 FACC

FACC is a real array, with one element for each axis in the system, and is used for defining the feedback acceleration value of the axis.

B.2.7 Routing Variables

The following are used for routing a feedback to a desired axis.



When using these variables the feedback sensors must relate to the same Servo Processor.

B.2.7.1 SLPROUT

SLPROUT is a real array, with one element for each axis in the system, and is used for setting the feedback routing of the position for the specified axis.

Its format is:

SLPROUT(axis_index) = value

Where value specifies the feedback source and can be one of those given in [Table 16-1](#).

Table 16-1. SLPROUT Values

Value	FPOS
0	According to E_TYPE position
001	From channel 0 Quadrature position or Absolute Encoder position
002	From channel 0 SINCOS position
003	From channel 0 HSSI position
101	From channel 1 Quadrature position or Absolute Encoder position
102	From channel 1 SINCOS position
103	From channel 1 HSSI position
201	From channel 2 Quadrature position or Absolute Encoder position
202	From channel 2 SINCOS position
203	From channel 2 HSSI position
301	From channel 3 Quadrature position or Absolute Encoder position
302	From channel 3 SINCOS position
303	From channel 3 HSSI position
X04	Rout from analog input X (where X can be 01 or 02)

The controller supports a standard control loop configuration where the 0 feedback position **FPOS** (0) is obtained from the 0 encoder, **FPOS**(1) from the 1 encoder, etc.

SLPROUT ≠ .0 indicates **FPOS** is from an alternative sensor, for example, if **SLPROUT**(0) is 0104, **FPOS** is obtained from an analog input 0 rather than from the encoder. In this case, the feedback source

could be a potentiometer or any other device that produces analog voltage proportional to the motor position.

The meaning of the routing value depends on the axis and the controller model. For example, a value of 1 specified for the 0 or 2 axis selects the 0 encoder, the same value for the 1 or 2 axis selects the 1 encoder.



For the values of the **E_TYPE** variable see the *SPiiPlus Command & Variable Reference Guide*.

B.2.7.2 SLVROUT

SLVROUT is a real array, with one element for each axis in the system, and is used for setting the feedback routing of the velocity for the specified axis.

Its format is:

SLVROUT(axis_index) = value

Where value specifies the feedback source that is routed to the FPOS variable and can be one of those in Table [Table 16-2](#).

Table 16-2. SLVROUT Values

Value	FPOS
0	According to E_TYPE velocity
001	From channel 0 Quadrature velocity or Absolute Encoder velocity
002	From channel 0 SINCOS velocity
003	From channel 0 HSSI velocity
101	From channel 1 Quadrature velocity or Absolute Encoder velocity
102	From channel 1 SINCOS velocity
103	From channel 1 HSSI velocity
201	From channel 2 Quadrature velocity or Absolute Encoder velocity
202	From channel 2 SINCOS velocity
203	From channel 2 HSSI velocity
301	From channel 3 Quadrature velocity or Absolute Encoder velocity
302	From channel 3 SINCOS velocity
303	From channel 3 HSSI velocity

Value	FPOS
X04	Rout from analog input X (where X can be 01 or 02)



For the values of the **E_TYPE** variable see the *SPiiPlus Command & Variable Reference Guide*.

B.2.7.3 SLCROUT

SLCROUT is an integer array, with one element for each axis in the system, and is used for setting the feedback routing of the velocity commutation for the specified axis.

Its format is:

SLCROUT(axis_index) = value

Where value specifies the feedback source that is routed to the FACC variable and can be one of those in Table [Table 16-3](#).

Table 16-3. SLCROUT Values

Value	FPOS
0	According to E_TYPE velocity
001	From channel 0 Quadrature velocity or Absolute Encoder velocity
002	From channel 0 SINCOS velocity
003	From channel 0 HSSI velocity
101	From channel 1 Quadrature velocity or Absolute Encoder velocity
102	From channel 1 SINCOS velocity
103	From channel 1 HSSI velocity
201	From channel 2 Quadrature velocity or Absolute Encoder velocity
202	From channel 2 SINCOS velocity
203	From channel 2 HSSI velocity
301	From channel 3 Quadrature velocity or Absolute Encoder velocity
302	From channel 3 SINCOS velocity
303	From channel 3 HSSI velocity
X04	Rout from analog input X (where X can be 01 or 02)



For the values of the **E_TYPE** variable see the *SPiiPlus Command & Variable Reference Guide*.

B.2.8 Configuring Dual Loop Control Procedure

This procedure is divided into two parts:

- > **AXIS Setup**

Where **AXIS** is the axis of the drive to which the motor is physically connected.

- > **LOAD Setup**

Where **LOAD** denotes the load feedback channel.

In general, the axis can be different from the motor feedback channel; however, this procedure assumes that the motor feedback is connected to the axis of the drive. Further it is assumed that the load's user units are employed with both feedback sensors, rather than counts. In this way, the Dual Loop setup is easier since fewer changes are needed (scaling gains, axis limits, and the like).

For purpose of illustrating the procedure, the following was used:

- > **Linear stage drive with a lead screw:**

A linear encoder for stage position, rotary encoder for motor velocity and commutation.

- > **User units: mm.**

- > **The motor is connected to axis '0'.**

- > **The motor feedback, connected to axis '0', is a rotary Quadrature encoder, with 2000 [lines/revolution].**

- > **The load feedback, connected to axis '1', is a linear Quadrature encoder with 12500 [lines/mm].**

- > **Both include an internal multiplier of 4.**

- > **The lead screw pitch: 1[inch/revolution]= 25.4[mm/revolution]**

B.2.8.1 AXIS Setup

For setting up the **AXIS** we employ the SPiiPlus MMI Application Studio **Adjuster Wizard**. The reason for this is that while the **Adjuster Wizard** cannot be used for Dual Loop control, it can easily be used for calculating the parameter values for a Single Loop thereby saving the user some time.

1. In the **Adjuster Wizard** Select Task, select the **Axis**.
2. Then for the task select **Setup New System or Controller**.

Click **Next**.

3. Fill in the details in the Initialization window, and click **Next**.
4. In the Axis Architecture: Axis Structure window setup **AXIS** as a single loop axis with the motor's feedback; the feedback topology should be **Single, on motor. Choosing User Units Applied To Load**. This is recommended for working in the same units with both feedbacks.
5. Proceed until the Axis Setup and Tuning step and skip Position and Velocity Loops tuning.

6. [Save to Flash](#) and exit **Adjuster Wizard**.
7. Calculate the user units for EFAC(AXIS).

User units applied to EFAC(AXIS), which defines the ratio between the motor feedback counts and the load user units, can be calculated as follows:

$$EFAC(AXIS) = \frac{1 \text{ load unit}}{\text{feedback resolution} \left[\frac{\text{lines}}{\text{mm}} \right] \cdot \text{multiplier} \cdot \frac{1}{\text{gear ratio} \left[\frac{\text{load units}}{\text{motor units}} \right]}}$$

For example, for a linear motor with a feedback resolution of 500[lines/mm] and an internal multiplier of 4, if the desired user units are [mm]:

$$EFAC = \frac{1}{500 \cdot 4} = 0.005 \Rightarrow 1 \text{ count} = 0.005 \text{ mm}$$

For a rotary motor connected to a ball screw:

- > Feedback resolution: 2000 lines/rotation
- > Gear ratio: 1 motor rotation = motion of load
- > Multiplier: 4
- > .Required user unit: .m (micron)

$$EFAC = \frac{1}{2000 \cdot 4 \cdot \frac{1}{300}} = 0.0625 \Leftrightarrow 1 \text{ motor count} = 0.0625 \mu\text{m}$$

B.2.8.2 LOAD Setup

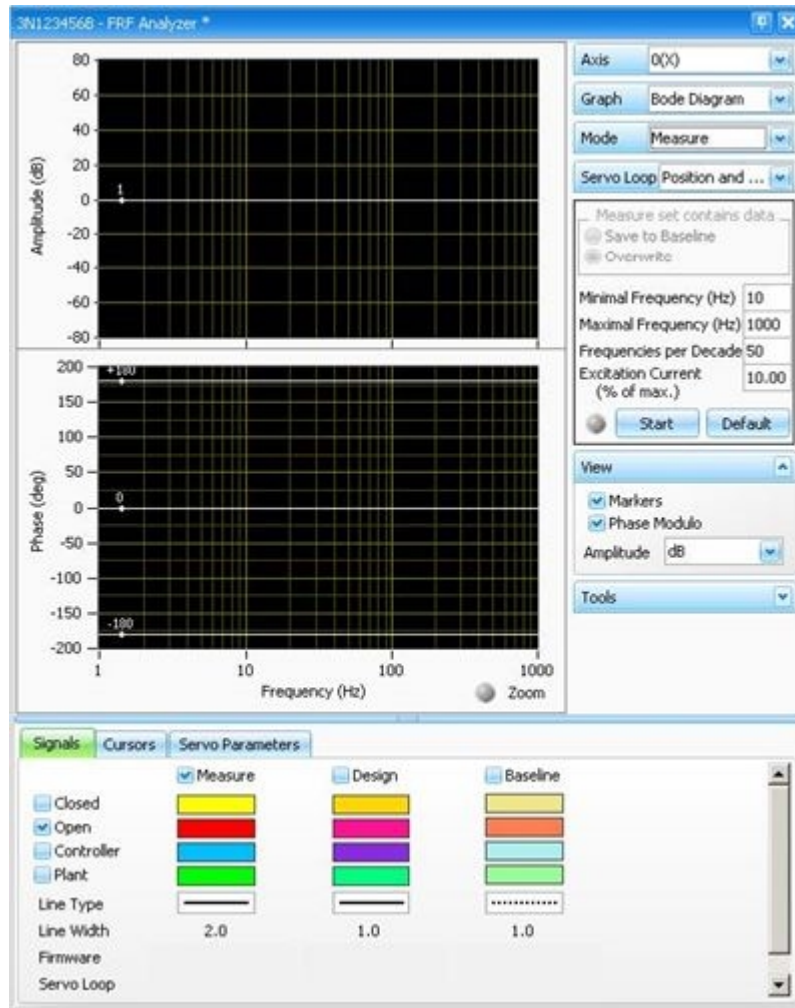
Using the value for EFAC(AXIS) calculated in [AXIS Setup](#), we now calculate the user units parameter for EFAC(LOAD).

The load feedback setup requires just setting EFAC(LOAD) and verifying that the load feedback functions as expected. (If the load feedback does not change, use the command **FCLEAR LOAD** to reset the load feedback).

We will be making use of the SPiiPlus MMI Application Studio FRF Analyzer. In particular, we will be making use of an improvement made to the FRF Analyzer in version 2.10:

In the Design mode, the **FRF Analyzer** simulates changes in the Position Loop servo parameter SLPKP and the Position Loop Bi-Quad filter parameters. A separate Position Loop measurement is required in Dual Loop systems.

1. In the SPiiPlus MMI Application Studio **Communication Terminal** enter the command: MFLAGS(AXIS).20=1 (this sets AXIS to Dual Control).
2. Open the SPiiPlus MMI Application Studio **FRF Analyzer** (for details of using the FRF Analyzer see the *SPiiPlus MMI Application Studio User Guide*):



3. Set the **FRF Analyzer** to the Design Mode.
4. Tune the Velocity Loop of the desired axis as in Single Loop. Note that the Bi-Quad filter in the Velocity Loop is disabled, as this filter has been moved to the Position Loop in Dual Loop mode (by the MFLAGS setting).



Velocity Loop tuning will not require changing if the same load user units are used.

5. Disable the axis. Set $SLVRAT(AXIS) = \frac{\text{position loop resolution}}{\text{velocity loop resolution}}$, using the feedbacks and gear parameters. SLVRAT can be calculated as follows:

$$SLVRAT(AXIS) = \frac{\text{position feedback's counts per user unit}}{\text{velocity feedback's counts per user unit}} = \frac{EFAC(LOAD)}{EFAC(AXIS)} = \frac{EFAC(AXIS)}{EFAC(LOAD)}$$

For example:

$$SLVRAT(0) = \frac{\text{position loop resolution}}{\text{velocity loop resolution}} = \frac{EFAC(0)}{EFAC(1)} = \frac{0.003175}{0.00002} = 158.75$$

6. Set the axis' encoder factor as the encoder factor of the load feedback: $EFAC(AXIS) = EFAC(LOAD)$



It is recommended initializing $FPOS(AXIS)$ and verifying that the new movement range does not reach the hard stop.

For example: $EFAC_{NEW}(0) = EFAC(1) = 0.00002$

7. To verify the value of $SLVRAT$, follow these steps (if manually moving the motor is possible):
 - > Zero both feedback positions.
 - > Disable the axis.
 - > Manually move the axis to an arbitrary position.
 - > Calculate $\frac{FPOS(LOAD)}{FPOS(AXIS)}$. This value should be close to the value calculated in Step 4.
8. Route the load feedback as the position feedback of the desired axis, by changing $SLPROUT(AXIS)$ as shown [SLPROUT Values](#).

For example, we want to route the load's Quadrature encoder from channel 1 to axis 0, so according to the table: **$SLPROUT(0) = 101$** .

9. If you need to read the motor feedback, it can be assigned to another axis, say Load, using $SLPROUT$ as above.



If **$XVEL$** was originally set according to the Load units, no further change is needed after step 6. Otherwise, **$XVEL$** should be changed according to **$SLVRAT$** before tuning the Velocity Loop, to maintain the Velocity Loop tuning.

10. Using the **FRF Analyzer** tune the Position Loop of the system in Dual Loop. Measure the Position Loop FRF separately in order to apply the Bi-Quad Filter, if needed.

Typical configurations of the Bi-Quad filter in the Position Loop are Notch filter or Lead filter which adds phase to the Open Loop FRF, allowing improved bandwidth and stability margins. See [Bi-Quad Filter](#) for a detailed explanation regarding the Bi-Quad filter tuning.

For verification, you may measure the Velocity Loop FRF separately. It should resemble the previous measurement of step 3.



When tuning the Position Loop, in order to produce a proper measurement and avoid system damage, remember to change the excitation level, since the excitation is a velocity command (% of **$XVEL$**), rather than current.

11. From this point, the Load Feedback is used for the Position Loop and the Motor Feedback is used for Commutation and the Velocity Loop.

Since the same load user units are used, XVEL(0) should not be changed and no further tuning is needed for the Velocity Loop:

$$XVEL_{\text{after step 5}} \left[\frac{\text{motor counts}}{\text{sec}} \right] = \frac{XVEL \left[\frac{\text{load units}}{\text{sec}} \right]}{EFAC(0)} = \frac{1270}{0.003175} = 40000$$

$$XVEL_{\text{after step 7}} \left[\frac{\text{motor counts}}{\text{sec}} \right] = \frac{XVEL \left[\frac{\text{load units}}{\text{sec}} \right]}{EFAC_{NFW}(0)} \cdot \frac{1}{SLVRAT} = \frac{1270}{0.00002} \cdot \frac{1}{158.75} = 400000$$

As can be seen, the raw value of $XVEL \left[\frac{\text{motor counts}}{\text{sec}} \right]$ remains the same after routing the load feedback to the Position Loop, so no further change in the Velocity Loop tuning is needed.

In the Position Loop tuning, the Bi-Quad was configured as a Notch filter and allowed the proportional gain to be further increased.

Appendix C. Working with Dynamic Braking

C.1 About Dynamic Braking

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 17-1.

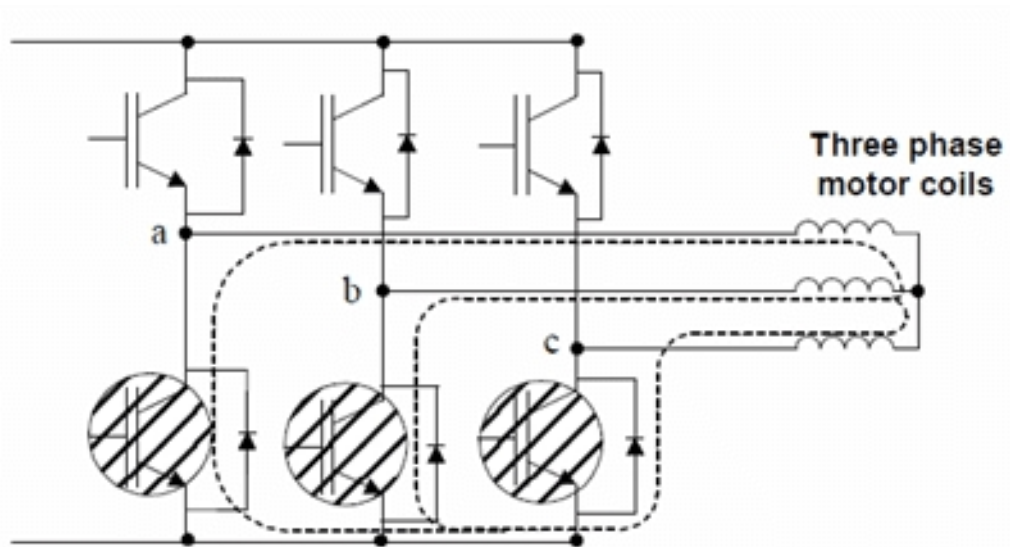


Figure 17-1. Dynamic Brake Implementation by the Digital Drive

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.



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:

SPIIPlus MMI Application Studio > Toolbox > Diagnostics and Monitoring > Scope

C.2 Dynamic Brake - Safety Verification

C.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.



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:

$$I_{DB_{max}} = \frac{K_E \cdot V_{max}}{\sqrt{3} \cdot |Z|} \quad (1)$$

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
$Z = R + j \times X, [\Omega]$	is the phase impedance (the sum of a phase resistance and phase reactance)

For a rotary motor:

$$I_{DB_{max}} = \frac{K_E \cdot \omega_{max}}{\sqrt{3} \cdot |Z|} \quad (2)$$

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
$Z = R + j \times X, [\Omega]$	is the phase impedance (the sum of a phase resistance and phase reactance)

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

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

where:

$L, [H]$	is the phase inductance
$\omega = 2 \times \pi \times f$	<p>f, [1/s] - is the frequency of the AC flowing through the circuit. f can be found from motor velocity as:</p> $f = \frac{V_{max}}{2 \cdot Pitch}$ <p>for the linear motor, where Pitch is 180 electrical degrees.</p>

$$f = \frac{\omega_{max}}{\frac{NP}{2}}$$

for the rotary motor, where NP is the number of poles.

The linear motor impedance is found from:

$$|Z| = \sqrt{R^2 + (\omega \cdot L)^2} = \sqrt{R^2 + (2 \cdot \pi \cdot f \cdot L)^2} = \sqrt{R^2 + \left(\pi \cdot \frac{V_{max}}{Pitch} \cdot L \right)^2} \quad (4)$$

The rotary impedance can be found from:

$$\begin{aligned} |Z| &= \sqrt{R^2 + (\omega \cdot L)^2} = \sqrt{R^2 + (2 \cdot \pi \cdot f \cdot L)^2} = \\ &= \sqrt{R^2 + \left(2 \cdot \pi \cdot \frac{\omega_{max}}{2 \cdot \pi \cdot (NP/2)} \cdot L \right)^2} = \sqrt{R^2 + \left(\frac{\omega_{max}}{(NP/2)} \cdot L \right)^2} \end{aligned} \quad (5)$$

Therefore, equation (1) can be represented as:

$$I_{DB_max} = \frac{K_E \cdot V_{max}}{\sqrt{3} \cdot \sqrt{R^2 + \left(\pi \cdot \frac{V_{max}}{Pitch} \cdot L \right)^2}} \quad (6)$$

Equation (2) can be represented as:

$$I_{DB_max} = \frac{K_E \cdot \omega_{max}}{\sqrt{3} \cdot \sqrt{R^2 + \left(2 \cdot \pi \cdot \frac{\omega_{max}}{(NP/2)} \cdot L \right)^2}} \quad (7)$$

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

Assum 2V drop on each transistor, the maximum allowed power dissipation on one transistor is:

$$2 \times I_{peak} \times 1sec$$

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:

$$\frac{1}{2}(J\omega^2 + Mv^2) - \Delta P \leq 6 \times I_{peak}$$

> In the case of a DC motor:

$$\frac{1}{2}(J\omega^2 + Mv^2) - \Delta P \leq 4 \times I_{peak}$$

where J [kgm²] 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.

C.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.

C.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:

- > For a DC brushless motor:

$$T_{max} = K_T I_{max} \frac{R}{Z} \approx K_T I_{max}$$

- > For a DC motor:

$$T_{max} = K_T I_{max}$$

where K_T is motor torque/ force constant.

C.2.4 Dynamic Brake - Verification Example

C.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 17-1](#).

Table 17-1. 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	0.4×10^{-3} [H]
Magnetic pitch (180 electrical degrees)	mm	15	60.96×10^{-3}

Kinetic energy (in Joule) is:

$$\frac{1}{2} \cdot 4 \cdot 1^2 = 2$$

This is lower than the permitted $6 \times 20 = 120$ 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 (in A):

$$I_{DB_{max}} = \frac{K_E \cdot V_{max}}{\sqrt{3} \cdot |Z|} = \frac{15 \cdot 1}{\sqrt{3} \cdot 1.2} = 7.2$$

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

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

The braking torque will be (in N):

$$\approx 17.5 \cdot \frac{7.2}{\sqrt{2}} = 89.1$$

C.2.4.2 Rotary Motor

A rotary motor application with maximum speed of:

$$\omega_{max} = 500000 \text{ [count/s]} = 500000/8000 \times 2\pi \text{ [rad/s]} = 392.7 \text{ [rad/s]} \text{ and } 20\text{A peak drive}$$

Three phase motor specifications are listed in [Table 17-2](#).

Table 17-2. 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 (in Joule):

$$\frac{1}{2} \cdot 0.000015 \cdot 392.7^2 = 1.15$$

This is lower than the $6 \times 20 = 120$ 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:

$$I_{DBmax} = \frac{K_E \cdot \omega_{max}}{\sqrt{3} \cdot |Z|} = \frac{\frac{16}{1000 \cdot 2 \cdot \pi} \cdot 60 \cdot 392.7}{\sqrt{3} \cdot 2.7} = 12.8$$

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

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

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

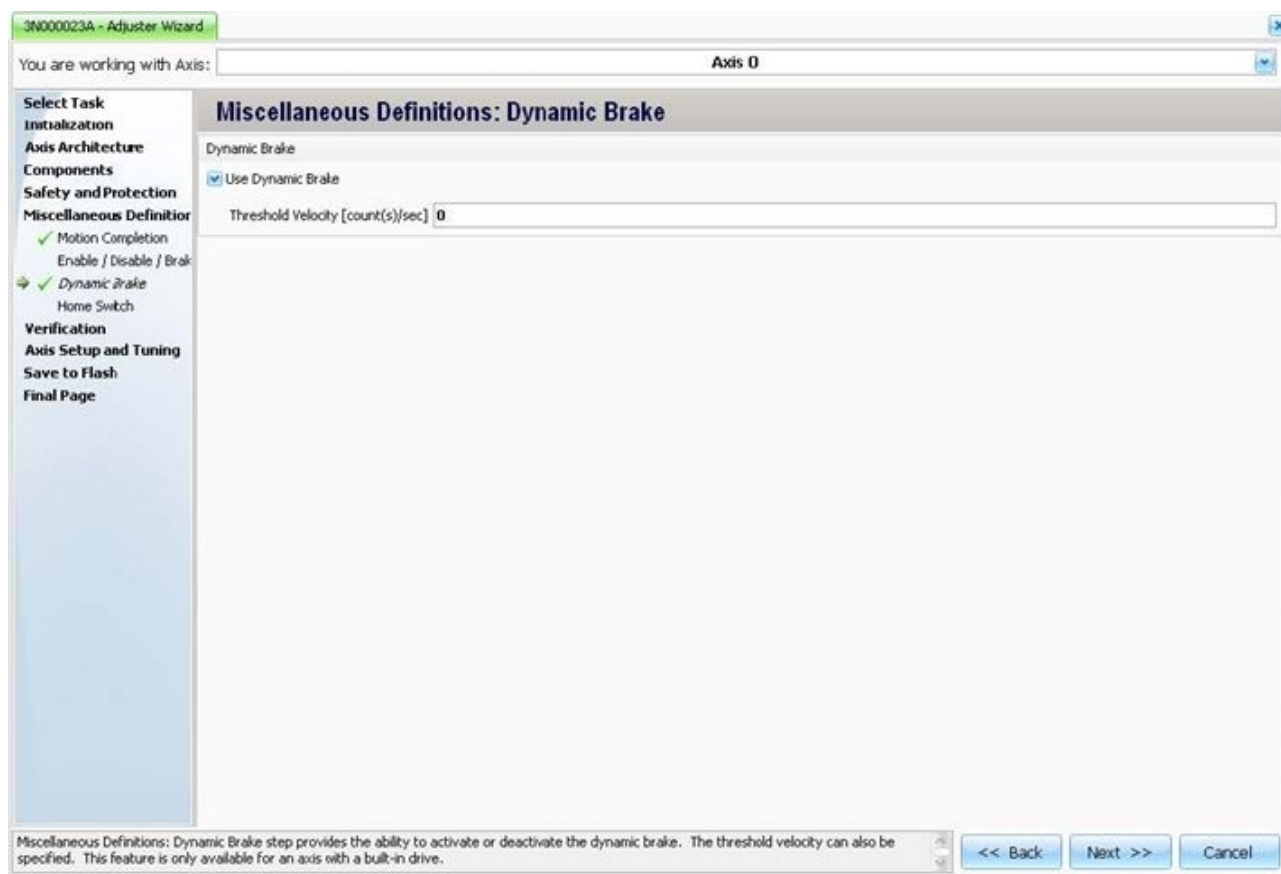
VELBRK should be: $1.5 \times 1000 \times 250 \times 256 = 96 \times 10^6$ [counts/sec]

C.2.5 Dynamic Brake Operation Procedure

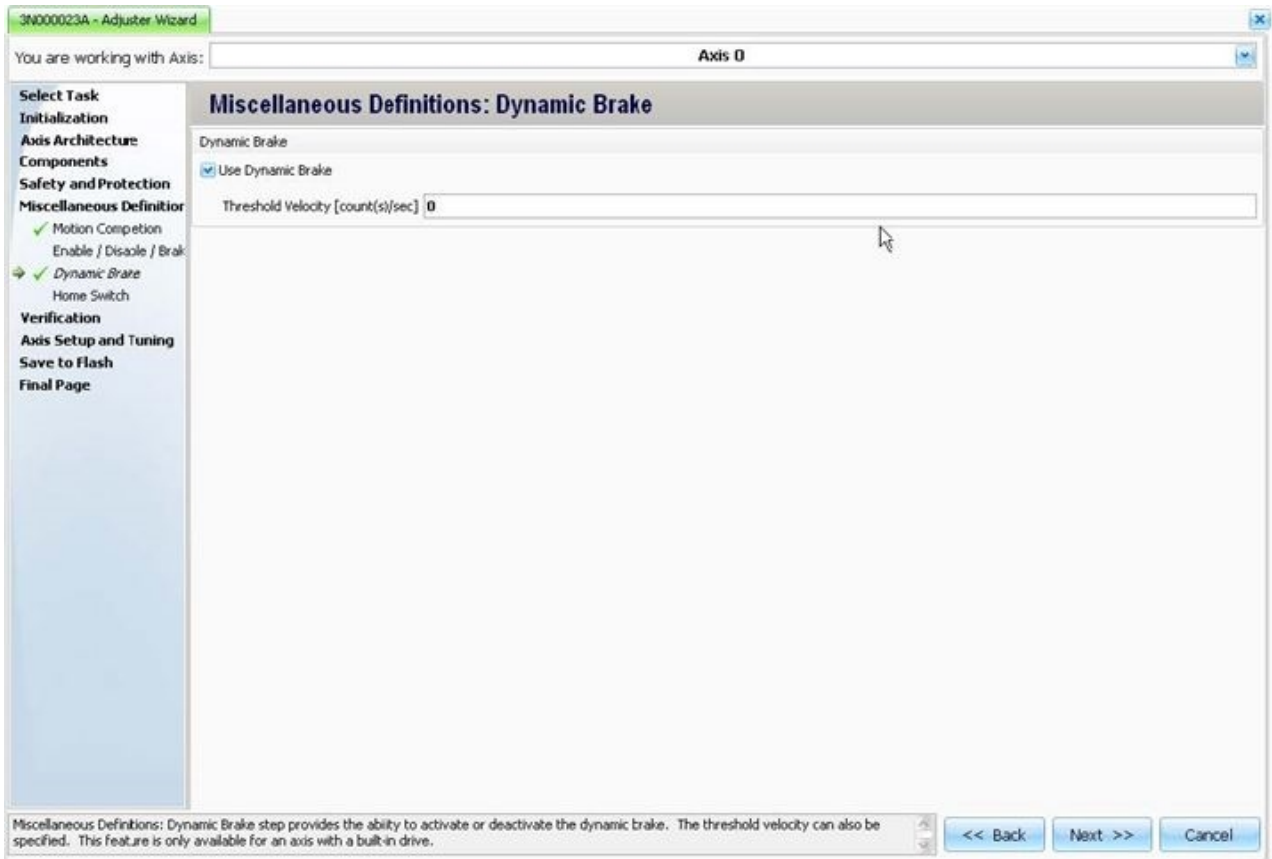


Before operating the Dynamic Brake, verify that no damage will happen due to the dynamic Brake.

You define the Dynamic Brake (**VELBRK**) value in the SPIIPlus MMI Application Studio **Adjuster Wizard** (see [About the Adjuster Wizard](#). **VELBRK** is one of the Miscellaneous Definitions task:



1. Select the **Use Dynamic Brake** checkbox.
2. Enter a value for the Dynamic Brake in the **Threshold Velocity (VELBRK)** field:



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

3. Click **Next** to continue with the **Adjuster Wizard**.

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

Appendix D. ASDF Measuring Motion Performance

D.1 Measuring Settling Time

1. Define the target position by assigning values to **SETTLE** and **TARGRAD** variables, see [Motion Completion](#).
2. Open **SPiiPlus MMI Application Studio > Toolbox > Diagnostics and Monitoring > Scope**. [Figure 18-1](#) depicts a Scope configured to measure settling time for a nanomotion piezoceramic motor.



Figure 18-1. Measuring Settling Time (Piezoceramic Motor Example)



[Figure 18-1](#) is taken from the old version of the SPiiPlus MMI. The operations in this procedure are valid for the new version details of which can be found in the *SPiiPlus MMI Application Studio User Guide*.

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 = $X2 - X1$.

Smarter



Motion

5 HaTnufa St.
Yokne'am Illit 2066717
Israel
Tel: (+972) (4) 654 6440 Fax: (+972) (4) 654 6443

Contact us: sales@acsmotioncontrol.com | www.acsmotioncontrol.com

