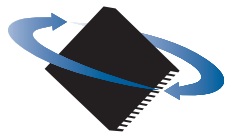
Juno™ Velocity & Torque Control IC
User Guide



PERFORMANCE
MOTION DEVICES

Performance Motion Devices, Inc.
1 Technology Park Drive
Westford, MA 01886

Revision 1.3 April, 2020

NOTICE

Warranty

Performance Motion Devices, Inc. warrants that its products shall substantially comply with the specifications applicable at the time of sale, provided that this warranty does not extend to any use of any Performance Motion Devices, Inc. product in an Unauthorized Application (as defined below). Except as specifically provided in this paragraph, each Performance Motion Devices, Inc. product is provided "as is" and without warranty of any type, including without limitation implied warranties of merchantability and fitness for any particular purpose.

Performance Motion Devices, Inc. reserves the right to modify its products, and to discontinue any product or service, without notice and advises customers to obtain the latest version of relevant information (including without limitation product specifications) before placing orders to verify the performance capabilities of the products being purchased. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement and limitation of liability.

Unauthorized Applications

Performance Motion Devices, Inc. products are not designed, approved or warranted for use in any application where failure of the Performance Motion Devices, Inc. product could result in death, personal injury or significant property or environmental damage (each, an "Unauthorized Application"). By way of example and not limitation, a life support system, an aircraft control system and a motor vehicle control system would all be considered "Unauthorized Applications" and use of a Performance Motion Devices, Inc. product in such a system would not be warranted or approved by Performance Motion Devices, Inc..

By using any Performance Motion Devices, Inc. product in connection with an Unauthorized Application, the customer agrees to defend, indemnify and hold harmless Performance Motion Devices, Inc., its officers, directors, employees and agents, from and against any and all claims, losses, liabilities, damages, costs and expenses, including without limitation reasonable attorneys' fees, (collectively, "Damages") arising out of or relating to such use, including without limitation any Damages arising out of the failure of the Performance Motion Devices, Inc. product to conform to specifications.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent procedural hazards.

Disclaimer

Performance Motion Devices, Inc. assumes no liability for applications assistance or customer product design. Performance Motion Devices, Inc. does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of Performance Motion Devices, Inc. covering or relating to any combination, machine, or process in which such products or services might be or are used. Performance Motion Devices, Inc.'s publication of information regarding any third party's products or services does not constitute Performance Motion Devices, Inc.'s approval, warranty or endorsement thereof.

Patents

Performance Motion Devices, Inc. may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.

Patents and/or pending patent applications of Performance Motion Devices, Inc. are listed at https://www.pmdcorp.com/company/patents.

Related Documents

Juno Velocity & Torque Control IC Programming Reference

Description of all Juno family IC commands, with coding syntax and examples, listed alphabetically for quick reference.

MC78113 Electrical Specifications

Complete electrical specifications for all Juno Velocity & Torque Control ICs. Contains physical and electrical characteristics, timing diagrams, pinouts, and pin descriptions.

DK78113 Developer Kit User Manual

How to install and configure the DK78113 Developer Kit. This developer kit supports all 64-pin TQFP Juno ICs, P/Ns: MC71112, MC73112, MC71113, MC73113, MC74113, MC75113, and MC78113.

Juno Torque Control IC User Guide

Complete description of the MC71112, MC71112N, MC73112 and MC73112N Juno torque control ICs including electrical characteristics, pin descriptions, and theory of operations.

DK73112N Developer Kit User Manual

How to install and configure the DK73112N Developer Kit. This developer kit supports the 56-pin VQFN Juno torque control ICs, P/Ns: MC71112N and MC73112N.

Juno Step Motor Control IC User Guide

Complete description of the MC74113, MC74113N, MC75113 and MC75113N step motor control ICs including electrical characteristics, pin descriptions, and theory of operations.

DK74113N Developer Kit User Manual

How to install and configure the DK74113N Developer Kit. This developer kit supports the 56-pin VQFN Juno step motor control ICs, P/Ns: MC74113N and MC75113N.

Table of Contents

List of Figures

This page intentionally left blank.

1. The Juno IC Family

In This Chapter

1.1 Introduction

This manual describes the Juno family of velocity & torque control ICs from Performance Motion Devices, Inc., consisting of the MC71113, MC73113, and MC78113 ICs for velocity control of Brushless DC, DC Brush, and step motors, the MC74113, MC74113N, MC75113, and MC75113N ICs for control of step motors, and the MC71112, MC71112N, MC73112, and MC73112N ICs for torque control of Brushless DC and DC Brush motors.

PMD's Juno ICs are ideal for a wide range of applications including precision liquid pumping, laboratory automation, scientific automation, flow rate control, pressure control, high speed spindle control, and many other robotic, scientific, and industrial applications.

The Juno family provides full four quadrant motor control and directly inputs quadrature encoder, index, and Hall sensor signals. It interfaces to external bridge-type switching amplifiers utilizing PMD's proprietary current and switch signal technology for ultra smooth, ultra quiet motor operation.

Juno ICs can be pre-configured via NVRAM for auto power-up initialization and standalone operation with SPI (Serial Peripheral Interface), direct analog, or pulse & direction command input. Alternatively Juno can interface via SPI, point-to-point serial, multi-drop serial, or CANbus to a host microprocessor.

Internal profile generation provides acceleration and deceleration to a commanded velocity with 32-bit precision. Additional Juno features include performance trace, programmable event actions, FOC (field oriented control), microstep signal generation, and external shunt resistor control.

All Juno ICs are available in 64-pin TQFPs (Thin Quad Flat Packages) measuring 12.0 mm x 12.0 mm including leads. The step motor control ICs and torque control ICs are also available in 56-pin VQFN (Very thin Quad Flat Non-leaded) packages measuring 7.2 mm x 7.2 mm. These VQFN parts are denoted via a "N" suffix in the part number.

1.2 Family Overview

The following table summarizes the operating modes and control interfaces supported by the Juno IC family:

Note that the MC78113 IC allows the motor type to be selected by the user. It provides all of the operating modes indicated for the MC71113, MC73113, and MC74113 Juno ICs.

	MC74113 MC74113N MC75113 MC75113N MC78113	MC71112 MC71112N	MC71113 MC78113	MC73112 MC73112N	MC73113 MC78113
Motor Type & Control Mode					
Motor Type	Step motor	DC Brush	DC Brush	Brushless DC	Brushless DC
Velocity			✓		✓
Torque/current	✓	✓	✓	✓	✓
Position & outer loop			✓		✓
Host Interface					
Serial point-to-point	✓	✓	✓	✓	✓
Serial multi-drop			✓		✓
SPI			✓		✓
CANbus			✓		✓
Command Input					
Analog velocity or torque		✓	✓	✓	✓
SPI velocity or torque		✓	✓	✓	✓
Pulse & direction	✓		✓		✓
SPI position increment	✓				✓
Motion I/O					
Quadrature encoder input	✓ (MC74113 & MC74113N only)		✓	✓	✓
Hall sensor input				✓	✓
Tachometer input			✓		✓
AtRest input	✓				
FaultOut output	✓	✓	✓	✓	✓
HostInterrupt output	✓	✓	✓	✓	✓
Amplifier Control					
PWM High/Low	✓	✓	✓	✓	✓
PWM Sign/Magnitude	✓	✓	✓		
DC Bus & Safety					
Shunt		✓	✓	✓	✓
Overcurrent detect	✓	✓	✓	✓	✓
Over/undervoltage detect	✓	✓	✓	✓	✓
Temperature input	✓	✓	✓	✓	✓
Brake	✓	✓	✓	✓	✓

1.3 Juno IC Developer Kits

Three different Juno developer kits are available. All of the 64-pin TQFP package Juno ICs are supported via the DK78113 developer kit board. The DK part numbers differ in the specific type of Juno IC that is installed.

Developer Kit P/N	Juno IC Installed	Motor supported	Comments
DK71112	MC71112	DC Brush	Torque control
DK71113	MC71113	DC Brush	Velocity & torque control
DK73112	MC73112	Brushless DC	Torque control
DK73113	MC73113	Brushless DC	Velocity & torque control
DK74113	MC74113	Step Motor	Provides quadrature encoder input
DK75113	MC75113	Step Motor	No quadrature encoder input
DK78113	MC78113	Multi-motor (Brushless DC, DC Brush, Step Motor)	Velocity & torque control with user-settable motor type
DK78113S	MC78113	Multi-motor (Brushless DC, DC Brush, Step Motor)	Socketed version of DK78113

The 56-pin VQFN IC package step motor ICs are supported by the DK74113N developer kit board. The DK part numbers differ in the specific type of Juno IC that is installed.

Developer Kit P/N	Juno IC Installed	Motor Supported	Comments
DK74113N	MC74113N	Step Motor	Provides quadrature encoder input
DK75113N	MC75113N	Step Motor	No quadrature encoder input

The 56-pin VQFN IC package torque control ICs are supported by the DK73112N developer kit board. The DK part numbers differ in the specific type of Juno IC that is installed.

Developer Kit P/N	Juno IC Installed	Motor Supported	Comments
DK71112N	MC71112N	DC Brush	Torque control
DK73112N	MC73112N	Brushless DC	Torque control

Each developer kit includes:

- Standalone board with easy to access connectors for fast setup and testing
- Pro-Motion autotuning and axis wizard setup software
- Complete Juno manual PDFs
- Extensive application schematic examples

1.3.1 Pro-Motion AutoTuning and Setup Software

The figure below shows a typical Pro-Motion to Juno development kit connection setup.

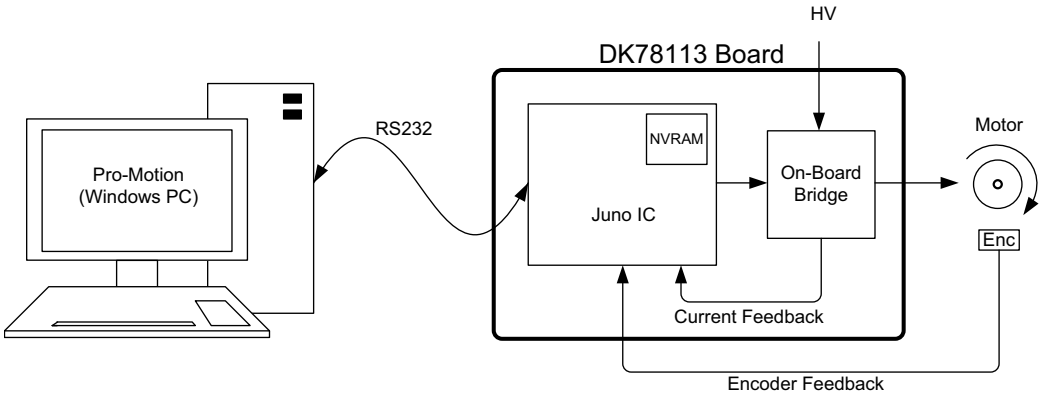


Figure 1-1: Juno Development Setup Connection Overview

PMD's Pro-Motion windows-based motion development system, which is included with all Juno developer kits, communicates directly to the Juno Developer Kit board. Pro-Motion provides numerous useful application development features including:

- Easy to use Axis Setup Wizard
- Motion trace display showing up to four simultaneously captured variables
- Frequency-based control optimization tools including Bode plot generator
- Motion development project save and retrieval
- Complete access to all Juno features and parameters
- NVRAM configuration

1.4 Guide to the Documentation

The Juno IC family is notable for the number of different control functions provided and motor types supported. While there are five different manuals specific to the Juno ICs, for a given application you will often need just one, or a smaller number of user guides. This is detailed in the table below:

Juno Function	Primary Reference	Companion References
Velocity Control	*Juno Velocity & Torque Control IC User Guide* is a superset description of the entire Juno IC family.	*MC78113 Electrical Specifications* *Juno Velocity & Torque Control IC Programming Reference.*
Torque Control	*Juno Torque Control IC User Guide* provides a convenient all-in-one reference for the Juno ICs that provide this dedicated control function.	*Juno Velocity & Torque Control IC User Guide* *Juno Velocity & Torque Control IC Programming Reference*
Step Motor Control	*Juno Step Motor Control IC User Guide* provides a convenient all-in-one reference for the Juno ICs that provide this dedicated control function.	*Juno Velocity & Torque Control IC User Guide* *Juno Velocity & Torque Control IC Programming Reference*

2. Functional Overview

2

In This Chapter

- Internal Block Diagram
- Signal Connections Overview
- Juno IC Operating Modes
- Control Flow Overview
- Typical Applications
- Juno Cycle Time & Loop Rates
- Motor Specific Versus Multi-Motor Juno ICs
- Host Commands
- Juno ICs in the Production Application

2.1 Internal Block Diagram

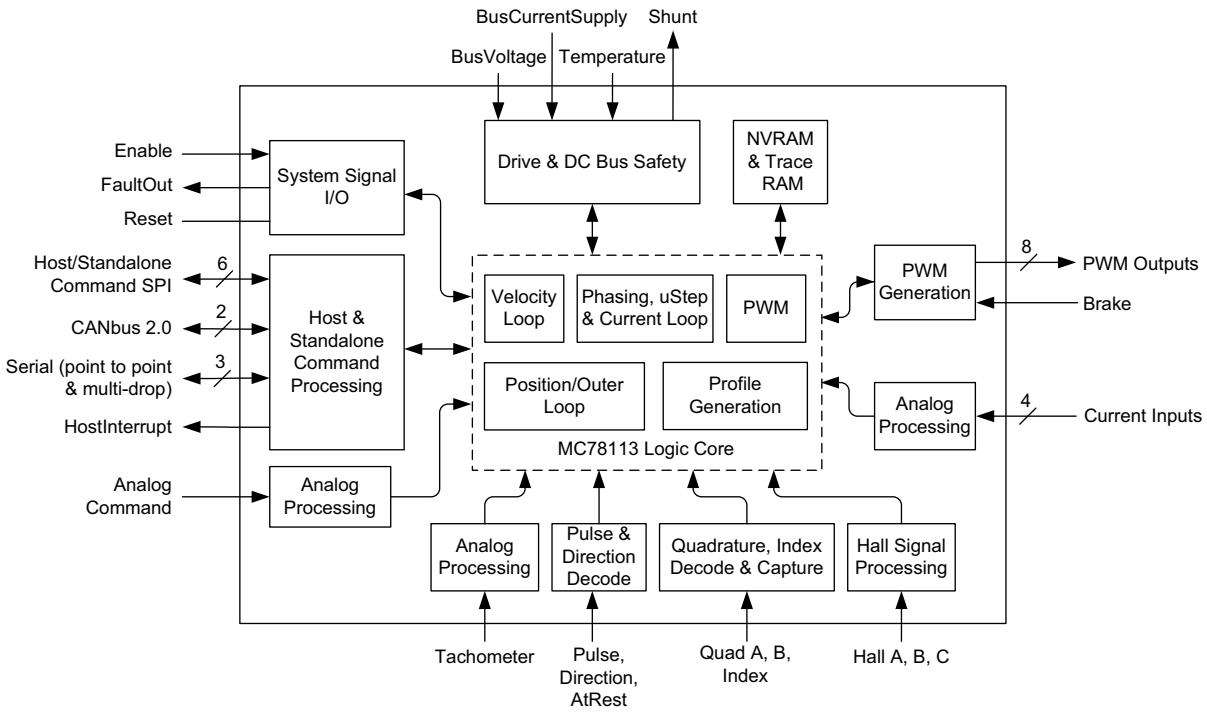


Figure 2-1: MC78113 Internal Block Diagram

Juno ICs are single-axis devices for velocity, torque, or voltage-mode control of three-phase brushless DC motors, DC Brush motors, or two-phase step motors. In addition Juno ICs can provide position control and control of "outer loop" quantities such as pressure and temperature when control of these quantities is related directly to the action of the motor.

At power-up or reset, Juno checks for the presence of stored configuration commands in its NVRAM. If NVRAM is programmed, the stored configuration commands are read into the chip, providing parameter information that will be used during operation. If no initial configuration is stored in NVRAM, then default values are used and information will then be sent by serial, CANbus, or SPI from a host device such as a microprocessor.

Depending on how the control loop has been configured an external analog signal may serve as the velocity or torque command value, an SPI (Serial Peripheral Interface) data stream may be used for the command value, or pulse & direction signals may provide a position command datastream. Alternatively an internal profile generator commanded by a host microprocessor via the serial, CANbus, or SPI communication port may be used to generate current, velocity, or outer loop command values.

Juno provides control of the motor position as well as outer loop quantities such as pressure or temperature via a PID (Proportional Integral Derivative) filter. Position control utilizes the incoming pulse & direction datastream to command step motor positions as well as Brushless DC or DC Brush motors. Outer loop control uses either the *Tachometer* analog signal or the digital SPI port to feedback the measured pressure or temperature.

Juno's velocity loop receives commands directly from analog or digital SPI circuitry or from the 'upstream' position/outer loop. The measured velocity may come from encoder, Hall sensor, or tachometer feedback. A PI loop, dual biquad filters, and a deadband filter allow a very wide range of precision velocity control applications to be addressed.

Current loop control is performed via direct input of analog signals representing the instantaneous current through the motor coils. These signals are typically derived from external dropping resistors or Hall sensors at the amplifier circuitry. This analog current information is then combined with the desired current for each phase to generate PWM signals.

To create a complete velocity or torque controller Juno is connected to switching amplifiers, typically MOSFET or IGBT-based. A programmable dead time function and other timing control parameters ensure that switch synchronization and control is optimal over the entire operating range of the driven motor.

A number of safety features are incorporated into the Juno ICs including I^2t current limiting, brake signal input, DC bus overvoltage and undervoltage detect, overcurrent detect, and overtemperature detect.

2.2 Signal Connections Overview

Figure 2-2 shows an overview of the connections used with Juno family ICs.

For additional information on Juno signals refer to the *MC78113 Electrical Specifications*.

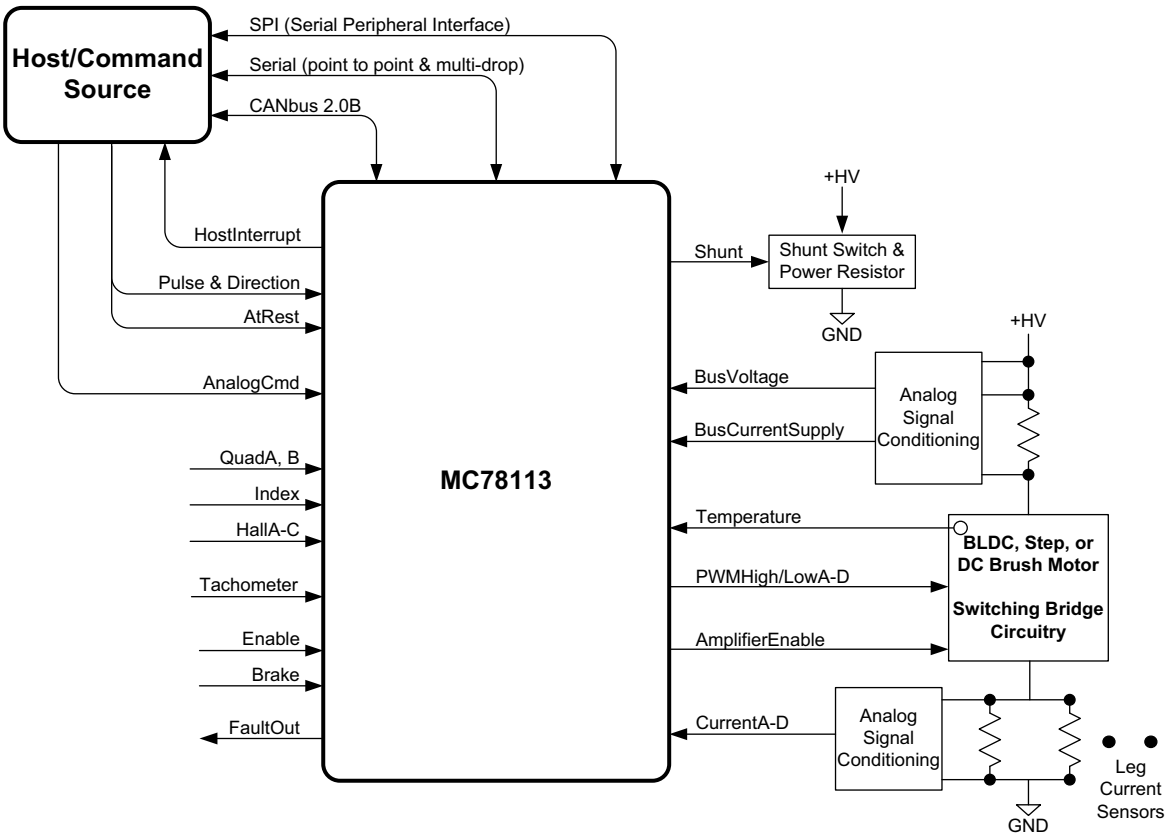


Figure 2-2: Juno IC Interconnections Overview

2.3 Juno IC Operating Modes

Juno can be used in one of two overall operating modes; as a standalone motor control IC driven by direct input command signals, or as an intelligent motor controller driven by a microprocessor or other controller sending high level host commands via serial, CANbus, or SPI.

2.3.1 Direct Command Operating Mode

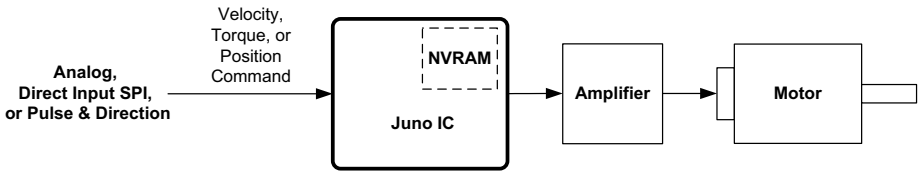


Figure 2-3: Direct Input Connections

When using Juno in direct command mode a continuous torque, velocity, position, or outer loop command is provided via external circuitry using analog, digital SPI, or pulse & direction signals. In this operating mode the configuration settings and gain parameters required by Juno are typically stored in advance into the internal NVRAM.

Upon powerup Juno reads the NVRAM, initializes itself according to this configuration information, and begins operation. See Chapter 12, *Power-up, Configuration Storage & NVRAM*, for more information on different options for storing data into the Juno's NVRAM.

2.3.2 Host Command Operating Mode

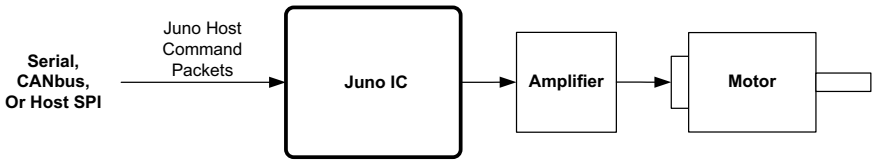


Figure 2-4: Host Command Connections

When using Juno ICs in host command mode a microprocessor or other programmable controller sends high level command packets. In this mode the NVRAM may still be used to store initialization information, but it is more common for the host microprocessor to send these initialization commands directly after powerup of the Juno IC.

The choice of direct input or host command operation of Juno is application specific. Even if direct input operation is used in the final application, host commands will at a minimum be used to program Juno's initialization NVRAM configuration.

2.4 Control Flow Overview

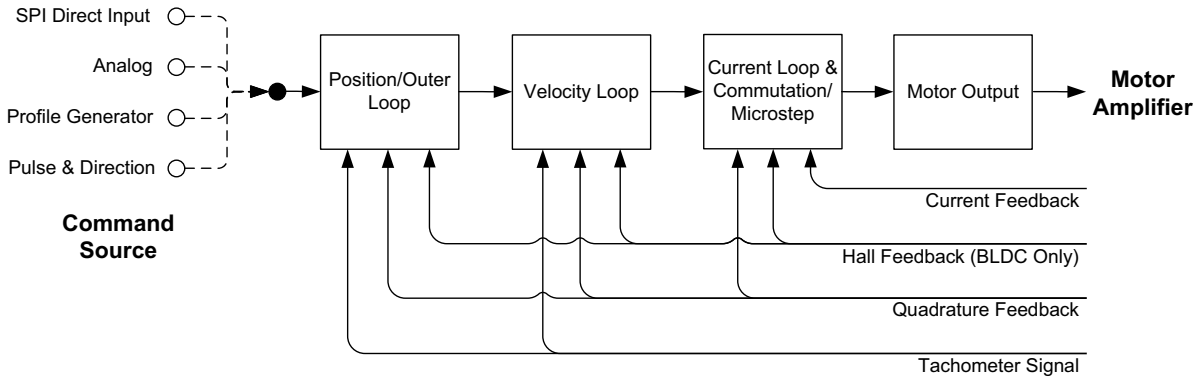


Figure 2-5: Juno Control Flow Overview

Figure 2-5 provides a control flow overview for the Juno IC. It shows how a final motor command is generated starting with the command source and ending with the motor output module that generates amplifier control output signals. Each of the major blocks within this control flow diagram is referred to as a loop control module.

The following table provides a brief description of each of the Juno loop control modules:

Module Name	Function
Position/Outer Loop	This module is used with servo motors only. It inputs the commanded position (the instantaneous desired axis position) and the actual position (the measured motor position), and passes the resultant position error (the difference between the commanded and the actual position) through a PID filter. When functioning as a pressure or temperature (outer loop) controller the functionality is similar but the commanded value is the outer loop value and the actual value is the measured outer loop value.
Velocity Loop	This module is used with servo motors only. It inputs a commanded velocity and an actual velocity to generate a velocity error which is then passed through a PI filter.

Module Name	Function
Current Loop	This module inputs the commanded current along with the actual current and uses a PI filter along with FOC (field oriented control) current control technology to generate motor voltage commands for each motor phase. For multi-phase motors such as Brushless DC and step motor this module also performs waveform generation.
Motor Output	This module inputs the motor phase commands and generates the appropriate electrical signals based on the selected electrical output format.

Depending on the type of motor used some modules may not be utilized. For example, step motors do not use a velocity loop. In addition, some modules may not be used for specific applications. For example the position/outer loop module and the velocity loop module are not used in torque-mode amplifier applications.

2.5 Typical Applications

Most Juno control applications fall into one of a few setup configurations in terms of which modules are enabled and which are disabled, what the input command sources are, and what the feedback sources are. A number of the most common setup configurations are summarized in the following sections.

2.5.1 Velocity Control of Brushless DC and DC Brush Motors

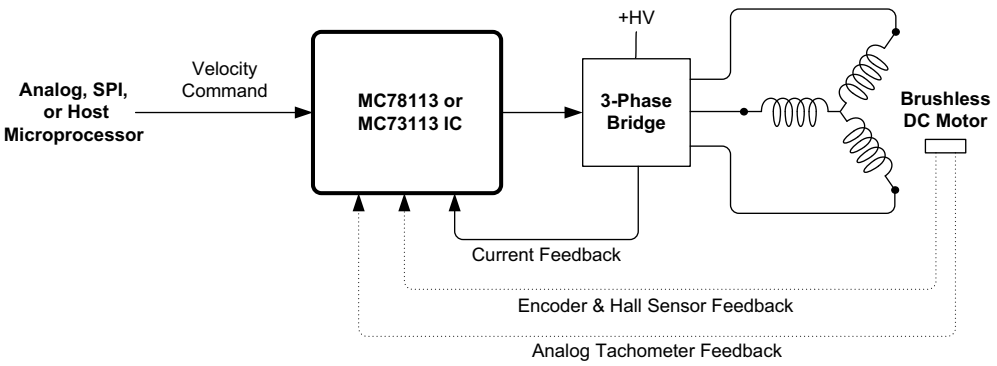


Figure 2-6: Servo Motor Velocity Control Diagram

Applications: General purpose amplifier, spindle control, centrifuge control, drug infusion, precision liquid pumping, turbine control.

In this configuration the Juno MC71113, MC73113, or MC78113 IC receives direct analog or direct input SPI (Serial Peripheral Interface) commands representing the instantaneous desired velocity, or host microprocessor commands representing the desired velocity profile. Quadrature encoder feedback, Hall sensors, or a tachometer are utilized for velocity feedback. For Brushless DC motors Hall sensors normally provide commutation feedback. If encoder signals are available however Halls are not required as long as the motor can move freely during startup (allowing PMD's pulse phase initialization to be used).

In the diagram above a Brushless DC motor is shown but similar velocity control can be provided for DC Brush motors. The diagram below shows the Juno control loop configuration for this application.

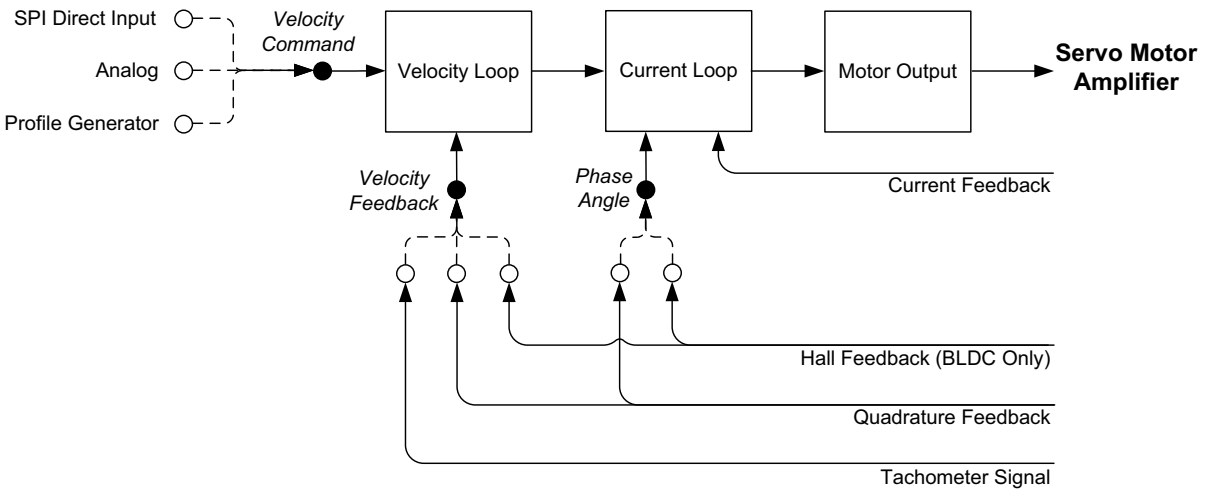


Figure 2-7: Servo Motor Velocity Control Loop Structure

2.5.2 Torque Control of Brushless DC and DC Brush Motors

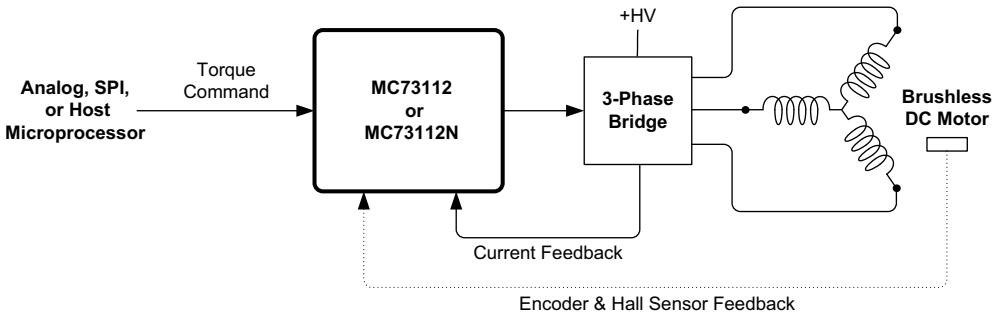


Figure 2-8: Servo Motor Torque Control Diagram

Applications: General purpose amplifier, spindle control, centrifuge control, drug infusion, precision liquid pumping, turbine control.

In this configuration the Juno MC71112, MC71112N, MC73112, or MC73112N IC receives direct analog or direct digital SPI (Serial Peripheral Interface) commands representing the instantaneous desired torque, or host microprocessor commands representing the desired torque profile. For Brushless DC motors Hall sensors normally provide commutation feedback. If encoder signals are available however Halls are not required as long as the motor can move freely during startup (allowing PMD's proprietary pulse phase initialization to be used).

The above diagram shows a Brushless DC motor but similar torque control can be provided for DC Brush motors. The diagram below shows the Juno control loop configuration for this application.

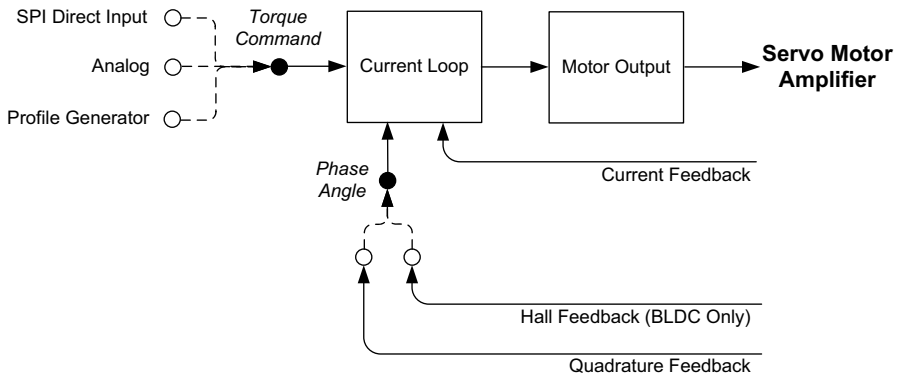


Figure 2-9: Servo Motor Torque Control Loop Structure

2.5.3 Pulse & Direction Control of Step Motors

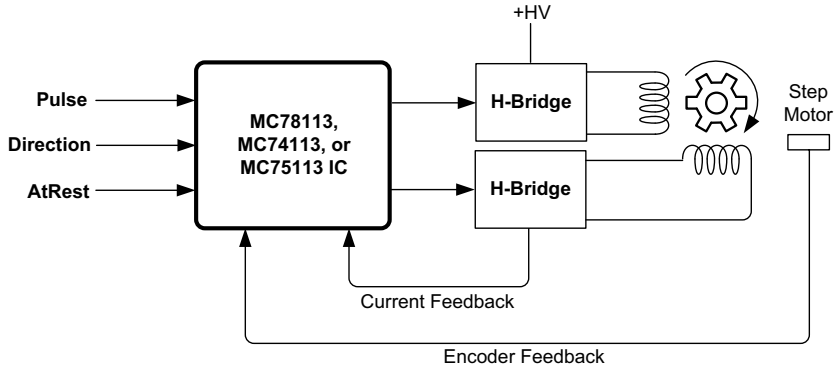


Figure 2-10: Step Motor Pulse & Direction Control Diagram

Applications: General purpose step motor drive, laboratory automation, liquid handling, scientific instruments, printers, XY stages.

In this configuration a microprocessor, PLC (programmable logic controller), dedicated motion control IC, or other external profile generator provides pulse, direction, and (optionally) at rest signal commands to the Juno MC74113, MC75113, or MC78113 IC.

The above diagram shows pulse & direction command input but SPI (Serial Peripheral Interface) incremental commands may also be used for position command input. The diagram below shows the Juno control loop configuration for this application.

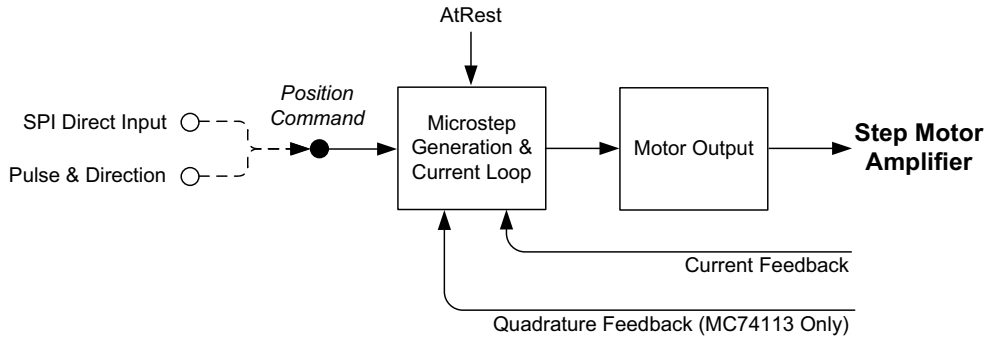


Figure 2-11: Step Motor Pulse & Direction Control Loop Structure

2.5.4 Pulse & Direction Control of Servo Motors

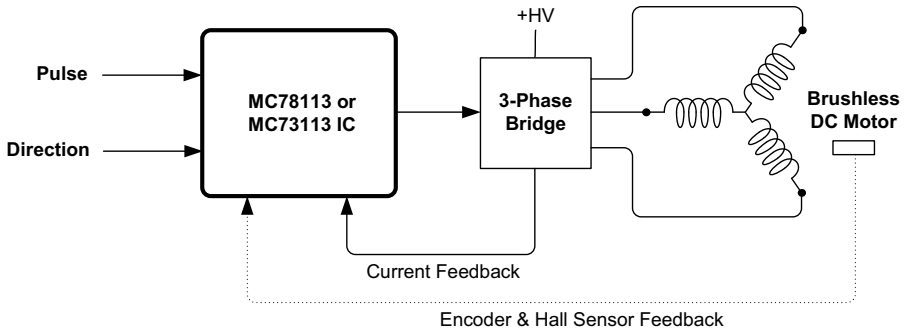


Figure 2-12: Pulse & Direction Control of a Servo Motor

Applications: General laboratory automation, liquid handling, scientific instruments, printers, XY stages.

In this configuration a microprocessor, PLC (programmable logic controller) dedicated motion control IC, or other external profile generator provides pulse and direction command signals to the Juno MC73113 or MC78113 IC.

Although a Brushless DC motor is shown in the diagram, a DC Brush motor may be similarly controlled. The diagram below shows the Juno control loop configuration for this application.

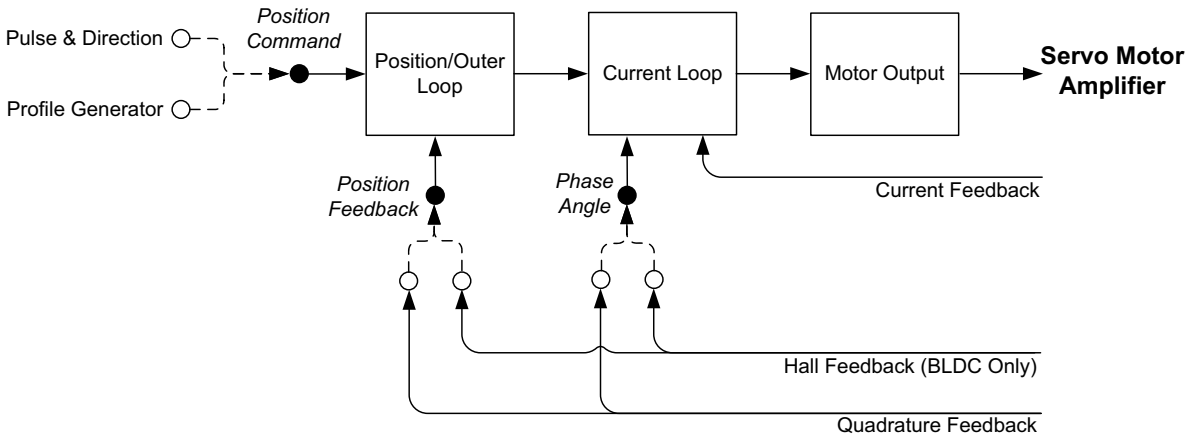


Figure 2-13: Position Mode Servo Amplifier Loop Structure

2.5.5 Pressure & Temperature Control with Servo Motors

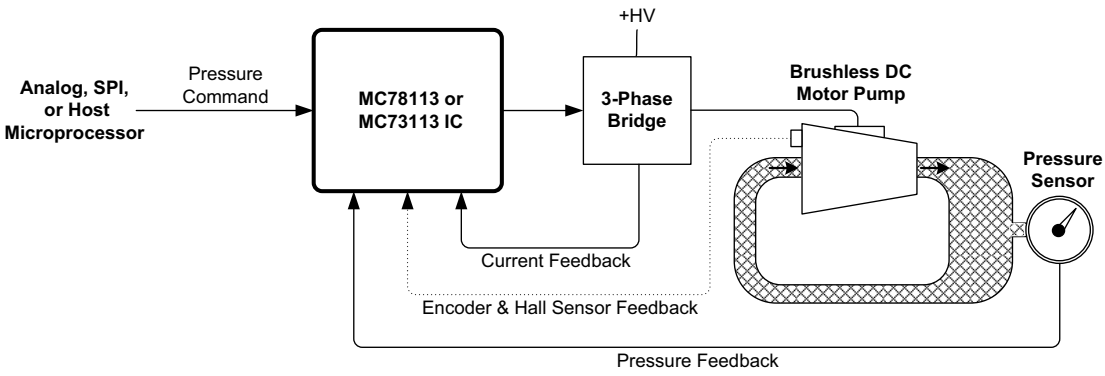


Figure 2-14: Pressure Control with a Servo Motor

Applications: Pressure control, flow control, temperature control, magnetic bearing control, galvanometer control, liquid level control.

In this configuration the Juno MC71113, MC73113, or MC78113 IC receives direct analog or direct digital SPI (Serial Peripheral Interface) commands representing the instantaneous desired pressure or temperature or host microprocessor commands representing the desired pressure or temperature profile. A pressure or temperature sensor provides an analog or direct input SPI feedback signal. Although a Brushless DC motor is shown in the diagram, a DC Brush motor may be similarly used.

Other types of "outer loop" control can be achieved with Juno as long as the measured quantity has a roughly linear relationship with the motor spin rate or driven actuator output. These controllable quantities/processes include pressure, temperature, flow rate, liquid level, magnetic bearing control, chemical reaction control, phase change control, and others.

The diagram below shows the Juno control loop configuration for this application.

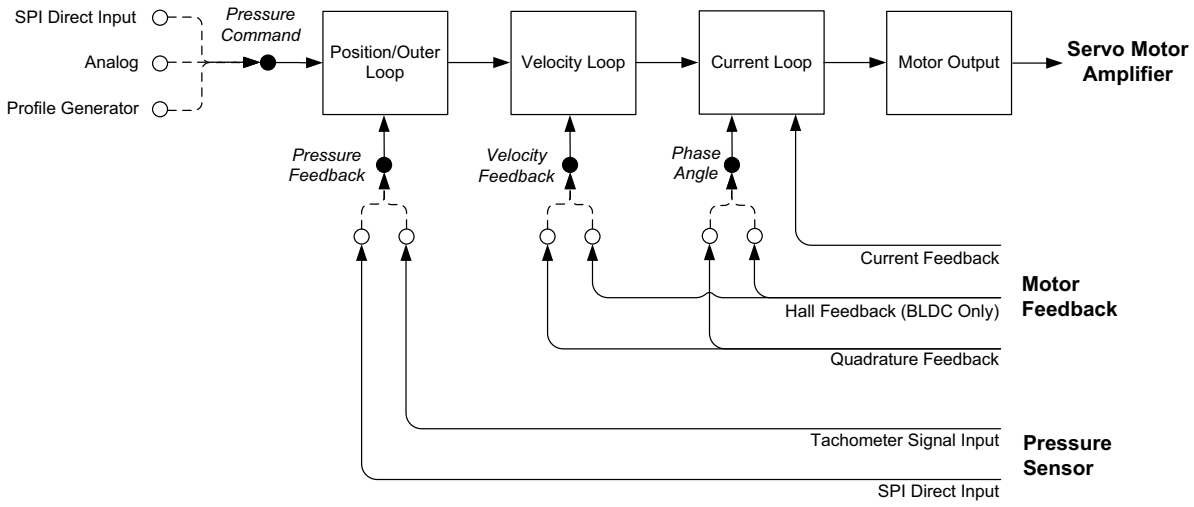


Figure 2-15: Outer Loop Controller Loop Structure

2.5.6 Torque Control with Magellan Motion Control IC Servo Applications

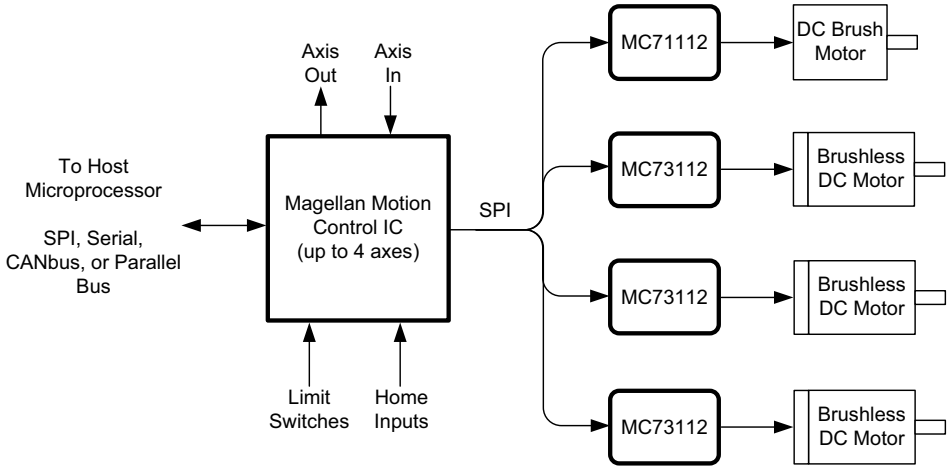


Figure 2-16: Torque Control with Magellan Motion Control IC Diagram

Applications: General purpose multi-axis motion control, laboratory automation, scientific instruments, XY stages, multi-dimensional contouring, semiconductor equipment.

PMD's Magellan Motion Control ICs provide up to four axes of profile generation, position servo loop control, pulse & direction signal generation, and numerous other synchronization and control features. In this application one or more Juno ICs connect to the Magellan IC and provide high performance current control and motor amplifier management for Brushless DC, or DC Brush motors.

In this configuration the Juno MC71112, MC71112N, MC73112, or MC73112N IC accepts a continually changing torque command via a SPI data stream, and drives the motor at those torque values using analog current feedback signals from the motor. The above diagram shows an SPI connection to the Juno ICs but a direct analog command input may be similarly used.

The diagram below shows the Juno control loop configuration for this application.

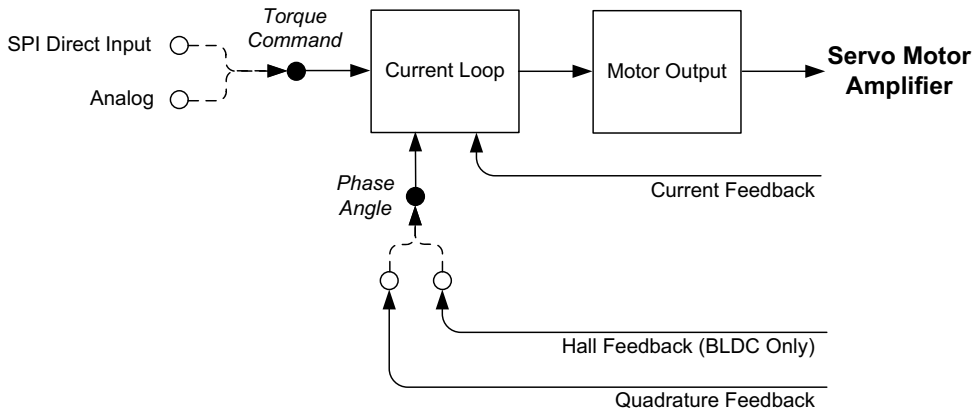


Figure 2-17: Magellan Connected Torque Control Loop Structure

2.6 Juno Cycle Time & Loop Rates

Juno ICs calculate all profile generator, position/outer loop, and velocity loop information on a fixed, regular interval. This interval is known as the cycle time. For Juno ICs the default as well as the minimum cycle time value is 102.4 µSeconds giving an update rate of approximately 10 kHz.

There are circumstances however where a slower loop rate is preferable. In particular, running at a lower loop rate may improve motion smoothness if the motor velocity is estimated from a quadrature encoder or from Hall signals. Ultimately the optimum cycle time is application specific and should be determined experimentally as part of the control parameters tuning process. To set the cycle time the command **SetSampleTime*** is used. To read back the current cycle time the command **GetSampleTime** is used.

In addition to setting the overall Juno cycle time, the Position/Outer loop, if used, may have its loop rate further reduced by setting the Outer loop period. See Section 3.3.5, "Outer Loop Period," for more information.

**Throughout this user guide command mnemonics are provided to illustrate how Juno is controlled through commands. These mnemonics are symbolic human-readable representations of the actual formatted command data packets that are sent during host communication or loaded from the NVRAM during startup. For more information refer to Section 2.8, "Host Commands."*

Note that Juno's current loop, commutation, microstep waveform generation, and motor output update rates are not affected by the user specified cycle time. For detailed information on these Juno functions refer to the *MC78113 Electrical Specifications*.

2.7 Motor Specific Versus Multi-Motor Juno ICs

One of the Juno family's more unique characteristics is that a user programmable motor type IC is available. Most Juno ICs control a specific motor type, either Brushless DC, step motor, or DC Brush. The MC78113 IC however allows the motor type to be programmed by the user and is known as the multi-motor version.

While motor-specific Junos are often used in applications where the motor type is known and fixed, in products where different motor types may be used the multi-motor Juno can provide a number of advantages. These include reducing the number of separate motor controller design projects, requiring just one IC type to be purchased, and reducing the number of different board types needed in inventory.

As detailed in the *MC78113 Electrical Specifications* it is not difficult to create a multi-motor amplifier design on a single board which supports all three motor types (or two of the three types).Whether or not a dedicated-motor Juno IC or the multi-motor Juno IC is used is application specific and up to the user to determine.

2.7.1 Setting the Motor Type

If the multi-motor MC78113 IC is used the command **SetMotorType** must be sent to specify the motor type. Either a DC Brush, Brushless DC, or step motor type is selected. The current motor type setting can be read using the **GetMotorType** command. If one of the motor specific Juno ICs is used it is not necessary to set the motor type.

Setting the motor type results in virtually all of Juno's control parameters being reset, and therefore the **SetMotorType** command should always be sent at the very beginning of the user's initialization sequence. This is true whether commands are sent via microprocessor host or via an initialization sequence stored in Juno's NVRAM.

2.8 Host Commands

Juno ICs provide more than 40 commands used for tailoring its control to a specific application. Depending on the specific Juno IC used, a serial, SPI, or CANbus, host port may be used to send these commands. Alternatively the NVRAM may be used to auto-load commands upon power-up. The format of the stored NVRAM commands closely resembles the format of commands sent via one of the host ports.

Most host commands specify a single parameter but some specify two or even three. Parameters may be signed or unsigned integers, may be bit encoded, or may be a fixed code specifying one of a list of available values. Throughout this manual we will show the mnemonics associated with Juno host commands.

Below are examples of host command mnemonics showing the variable name, associated mnemonic code, and range of settable values:

Parameter	Host Command Mnemonic	Range & Description
PWM Switching Frequency	20 kHz 40 kHz 80 kHz 120 kHz	Higher inductance motors should be set for 20 kHz. Lower inductance motors may use 40, 80, or 120 kHz to maximize current control accuracy and minimize heat generation. The default value for this parameter is 20 kHz.
Encoder to step ratio	SetEncoderToStepRatio	Two specified values, each have a range 1 to 32,767. The first parameter sets the number of encoder counts per motor rotation, the second specifies the number of microsteps per motor rotation

For a complete description of each command supported by Juno refer to the *Juno Velocity & Torque Control IC Programming Reference*.

2.8.1 Command Script Files

Figure 2-18: Sample Pro-Motion Script File

```
#ScriptVersion 1
:DESC "Motor 2 settings"
:CVER 1.3
SetDrivePWM 1 561
SetDrivePWM 2 0x80ff
SetDrivePWM 4 8
SetDrivePWM 5 2013
SetDrivePWM 6 2013
SetOutputMode 7
SetMotorCommand 0
SetSignalSense 0x0001
SetPhaseParameter 0 0
SetCurrentControlMode 1
SetFOC 512 680
ETC...
```

Juno ICs process host commands in their native 'machine' packet format consisting of a series of hexadecimal numbers. When used to record commands that will be stored in NVRAM during initialization however, PMD's Windows-based Pro-Motion program can use a special text file format to store host commands. This file is known as a script file and an example is shown in Figure 2-18. Script files are convenient because they are human readable and editable.

Script files are parsed via a specific syntax format. For more information refer to the *Juno Velocity & Torque Control IC Programming Reference*.

The primary purpose of script files and Juno's NVRAM power-up command execution scheme is to automatically load control parameters prior to starting motor control. Although Juno's NVRAM initialization sequence allows some synchronization with external signals and some timing control, it does not provide general purpose language controls such as looping or conditional code execution. If this type of control is desired than a microprocessor commanding the Juno IC via one of the host communication ports should be used.

Most users will not directly edit the script files and will instead rely on Pro-Motion to create the script file and store configuration information. Pro-Motion has convenient features for exporting and importing the current Juno configuration to a script file, or loading or uploading previously stored data from the Juno IC's NVRAM.

2.9 Juno ICs in the Production Application

Each Juno IC, before undertaking velocity control, torque control, or step motor control, must be programmed with parameter settings appropriate for the application that it will be used in. These control parameters include quantities such as PWM (Pulse Width Modulation) frequency, current loop gains, safety thresholds, and more.

One option for loading these parameters and actively controlling the Juno IC during operation is via a microprocessor host port. However another useful option is by using the NVRAM. Production related aspects of loading Juno's NVRAM with content is discussed in the next several sections.

2.9.1 Loading the NVRAM

There are two primary ways that NVRAM data can be programmed into the Juno IC in the user production application. Which method is best depends on the Juno package type you are using (56-pin VQFN versus 64-pin TQFP), and the preferred method of PCB board production and setup.

2.9.1.1 NVRAM Programming via Juno DK IC Socket

The socketed version of the DK78113 Juno DK (P/N DK78113S) can be used to program the NVRAM on 64-pin TQFP Juno ICs prior to soldering into the user's production PCB. Pro-Motion as well as a more compact programming executive available from PMD supports script files to program the Juno IC NVRAM. For more information on PMD script files refer to Section 2.8.1, "Command Script Files."

The 56-pin VQFN Juno DKs do not have a socket and therefore cannot be used to program the NVRAM of production 56-pin VQFN Juno ICs.

2.9.1.2 NVRAM Programming Via 3-pin UART Cable

An alternate NVRAM programming approach is to execute the NVRAM download by communicating to the Juno IC after it is installed in the production PCB. This approach generally uses a 3-pin connector installed on the user's production board. A technician plugs into this connector and performs the NVRAM download. To be programmed the Juno IC must receive power, so this generally means the PCB power is turned on during this procedure.

To facilitate this approach PMD provides a dedicated USB to 3-pin UART programming cable (PMD Part # Cable-USB-3P) with each Juno DK. This programming cable plugs into the PC's USB port on one end and into a 1x3 3-pin 2 mm header component on the other. A representative PCB mounted header component is Samtec MTMM-103-04-x-S-150.

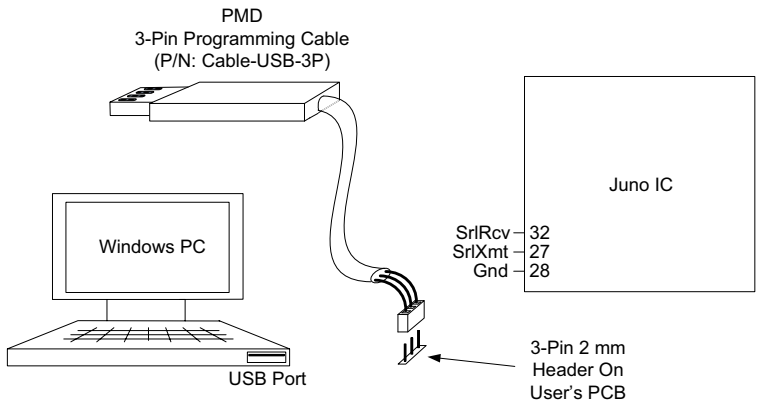


Figure 2-19: NVRAM Programming Via 3-pin UART Cable

The table below shows the required wiring for the on-board connector if it is to be used with the PMD programming cable:

Connector Pin #	Pin Name	Juno Pin # (64-pin TQFP Package)	Juno Pin # (56-pin VQFN Package)
1	SrlXmt	27	24
2	SrlRcv	32	28
3	GND	28 (or other ground signal)	25

2.9.2 Analog Signal Calibration in the Production Application

After integration into a PCB, it is recommended that the external analog signal processing circuitry that inputs to the Juno IC be calibrated. While some applications will not need these calibrations, for applications where the quietest, smoothest, and most accurate motion is desired, calibration of the analog inputs is recommended. Depending on the specific Juno IC used the signals that can be calibrated are *AnalogCmd*, *CurrentA-CurrentD*, and *Tachometer*. For more information on these signals refer to Chapter 14, *Hardware Signals.*

When a microprocessor is on the user PCB, generally this microprocessor is used to send the host commands needed to calibrate the analog inputs as part of the power up sequence.

Another approach is to calibrate during NVRAM-based initialization startup. This is done by embedding a command sequence that executes the calibration during the NVRAM startup at each power cycle. This approach can only be used when the startup condition of the PCB and connected motors is controllable. For example if the calibration occurs when the motors are still spinning, the calibration will not give accurate results.

A third approach that has the benefit of eliminating the need for calibration at each power cycle is to execute the calibration on the assembled PCB using a 3-pin UART programming cable. The derived calibration offsets are stored into NVRAM and recalled automatically thereafter by Juno at each power-up. For more information see Section 2.9.1.2, "NVRAM Programming Via 3-pin UART Cable."

PMD's Pro-Motion software program provides a convenient set up facility for selecting NVRAM startup options. In addition, advanced users can directly edit Juno start up scripts using a text editor. Refer to Section 2.8.1, "Command Script Files," for more information.

This page intentionally left blank.

3. Position/Outer Loop

3

In This Chapter

- Position Loop Operation
- Outer Loop Operation
- Settable Parameters
- Position/Outer Loop Calculations
- Motion Error Detection
- Watchdog Timer
- Position/Outer Loop Operation Startup
- Enabling and Disabling the Position/Outer Loop Module

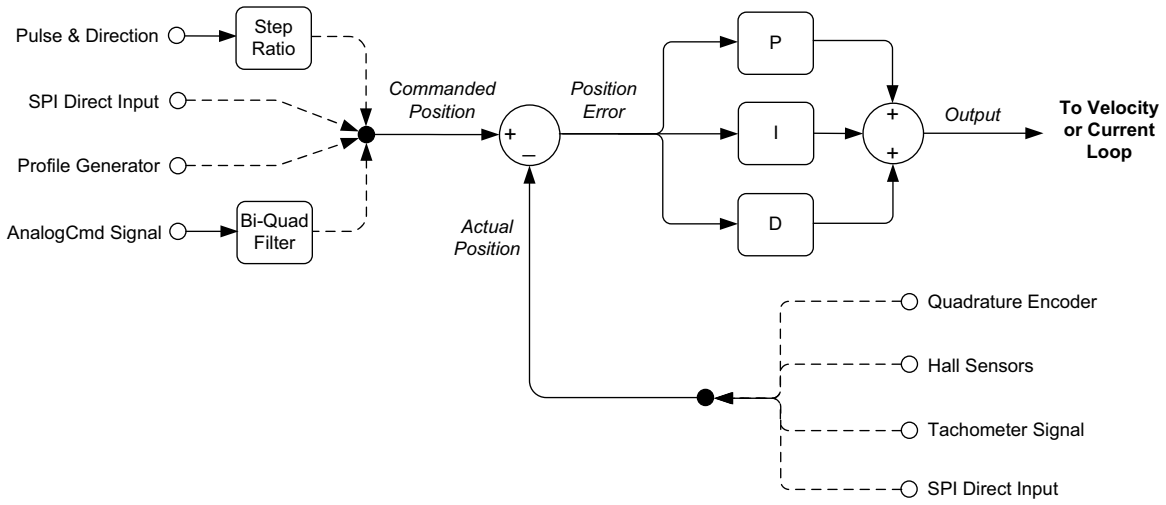


Figure 3-1: Position/Outer Loop Control Flow

Figure 3-1 provides a summary of the control flow of Juno's position/outer loop control module. The position/outer loop module, as the name suggests, is used for position control of the motor when the command source is a position rather than a velocity or a torque. In addition, this module can be operated as the outer loop controller for systems that control system characteristics such as pressure or temperature.

The Juno position/outer loop allows the user to specify one of several desired position or outer loop command sources as well as the corresponding measured value source. The loop command and the corresponding measured value are then subtracted to develop a loop error which is passed through a PID (proportional, integral, derivative) filter and output to the next enabled downstream control module, usually either the velocity loop or the current loop.

3.1 Position Loop Operation

The most common use of Juno's position/outer loop module as a position controller is when pulse & direction is selected as the command source. Incoming pulse & direction signals are counted and are stored in a 32 bit position register. This raw count is then scaled by a user specified ratio of steps to encoder counts resulting in a 32-bit loop command value.

An alternative command source in this mode is the position register of the profile generator. This is a common Juno operating mode for velocity control applications where the commanded axis moves very slowly, and where a discrete position based technique for velocity control may work better than an explicit velocity loop technique.

Finally, the direct input SPI port can be selected as a command source. The SPI port specifies a velocity command which is integrated to form a position command. Similar to the profile generator command source described above, this command source may be useful when the commanded axis moves very slowly.

To select the position loop command source the **SetDriveCommandMode** command is used. To read this value back the **GetDriveCommandMode** command is used. To read the current value of the pulse & direction position register the **GetCommandedPosition** command is used. To read the profile generation position register the **GetCommandedPosition** command is used.

There are two selectable position feedback sources; quadrature encoders and Hall sensors. Similar to the command sources, the measured feedback position is kept as a 32-bit register. To set the feedback source the command **SetEncoderSource** is used. To read this value back the **GetEncoderSource** command is used.

Although the velocity loop can be enabled when position control of the servo motor occurs (this is referred to as a cascaded position-velocity controller), more common is that the position loop is used with the velocity loop disabled and for the output of the position/outer loop module to directly command the current loop. For more information on current loop functioning refer to Chapter 5, *Current Loop*.

3.2 Outer Loop Operation

The term "outer loop" refers to the Juno capability to control measurable physical system characteristics when control of that characteristic has a roughly proportional response to the Juno-controlled motor velocity.

The most common use of this capability is to control pressure or temperature within a chamber. A sensor measures the pressure or temperature and the output of the outer loop is fed downstream to the velocity loop, which actively controls the spin rate of the motor. This in turn, due to the mechanical relationship of the motor to the measured pressure or temperature, increases or decreases the pressure or temperature.

Both positive and negative motor to measured characteristic relationships are supported. For example when controlling pressure, an increase in motor velocity typically increases the pressure. However when controlling temperature via a circulating cooling fluid, an increase in motor velocity decreases the temperature.

When the position/outer loop module is used for outer loop control the selectable command sources are direct input SPI, the *AnalogCmd* signal, and the profile generator velocity register. Both the direct input SPI command value and the analog command input value hold a signed 16 bit quantity representing the command value. If the profile generator is set as the source the value of the 32 bit velocity register will be used after being scaled by a user-specified velocity scalar value.

As shown in Figure 3-1, if selected as the command source the *AnalogCmd* signal is passed through a biquad filter. Although often left at its default filtering which consists of a low pass filter with a cutoff of 675 Hz, the biquad filter may be useful for applications where more sophisticated loop filtering is desired. For more information on Juno's biquad filters and how to set them, refer to Section 4.5, "Biquad Filtering."

To select the command source the **SetDriveCommandMode** command is used. To read this value back the **GetDriveCommandMode** command is used. To set and read back the velocity scalar the commands **SetLoop** and **GetLoop** are used. To read the current value of the direct input SPI register the command **GetLoopValue** is used. To read the current value of the *AnalogCmd* signal the command **GetLoopValue** is used. To read the current value of the velocity register the command **GetCommandedVelocity** is used.

There are two selectable feedback sources in this mode; the *Tachometer* analog input signal (providing the measured pressure or flow rate rather than the motor spin rate) and direct input SPI. To set the feedback source the **SetLoop**

command is used. To read this value back the **GetLoop** command is used. To read the 16-bit current value of the Tachometer signal the command **GetLoopValue** is used.

3.3 Settable Parameters

To control the position/outer loop up to ten parameters are set by the user; Kp, Ki, Kd, Ilimit, Dtime, Kvel, Kout, Outposlimit, Outneglimit, and Pouter. Three of these are gain factors for the PID (proportional, integral, derivative) controller, one is a limit for the integral contribution, one is the derivative sampling time, one is a scale factor that is applied only when the command source is set to profile generator, one is an output scale factor that is always applied, two are signed limits to the loop output value, and one is the cycle time period for the position/outer loop.

Term	Name	Default Value	Representation & Range
Kp	Proportional gain	0	Unsigned 16 bits (0 to 32,767)
Ki	Integral gain	0	Unsigned 16 bits (0 to 32,767)
Kd	Derivative gain	0	Unsigned 16 bits (0 to 32,767)
Ilimit	Integral limit	0	Unsigned 32 bits (0 to 2,147,483,647)
Dtime	Derivative time	1	Unsigned 16 bits (0 to 32,767 cycle times)
Kvel	Velocity scalar	65,536	Unsigned 32 bits (1 to 2,147,483,647)
Kout	Output scalar	32,767	Signed 16 bits (-32,768 to +32,767)
Outposlimit	Positive output limit	2,147,483,647	Unsigned 32 bits (0 to 2,147,483,647)
Outneglimit	Negative output limit	-2,147,483,648	Signed 32 bits (-2,147,483,648 to 0)
Pouter	Outer loop period	1	Unsigned 16 bits (1 to 32,767 cycle times)

All of the above parameters are set with the **SetLoop** command and read with the **GetLoop** command.

Determining correct parameters for the Kp, Ki, Kd, and Ilimit gains can be done in a number of ways. The easiest is to utilize the auto-tuning facility provided within PMD's Pro-Motion software package. Parameters derived using this procedure may or may not be optimized for your system but will be adequate for most applications and a good starting point.

> Please note that it is the responsibility of the user to determine the suitability of all control parameter values, including those determined by auto-tuning, for use in a given application.

In addition to these settable parameters there are settable parameters associated with the biquad filter, should it be changed from its default values. Refer to Section 4.5, "Biquad Filtering" for more information.

3.3.1 Derivative Time

Normally, the derivative time of the position or outer loop PID controller, called Dtime, is set so that the derivative contribution is recalculated at every servo cycle. Under some circumstances however it may be desirable to set the derivative sampling rate lower than this to reduce noise in the estimated derivative and to improve system stability or simplify tuning.

The specified value is the desired number of servo cycles per Juno cycle time. For example, if Juno's cycle time (set using the **SetSampleTime** command) has been set to 204.8 µSec, a value of 10 programmed in the derivative time register will result in a derivative sample time of 204.8 µSec * 10 = 2.048 mSec.

Changing the derivative sample time has no effect on the overall cycle time set using the command **SetSampleTime**.

The default value for the derivative time is 1, meaning that by default the derivative term is calculated at each servo cycle.

3.3.2 Velocity Scalar

The Juno velocity scalar register, called Kvel, is used by several control modules including the position/outer loop module and the velocity loop module. When used in conjunction with the outer loop control function it is used to scale the profile generator's 32-bit velocity register to generate the outer loop command.

Kvel is an unsigned 32 bit number with 1/65,536 scaling meaning that a scale factor of 1.0 (unity scaling) is expressed with a value of 65,536.

Example

In a pressure control application the current pressure of the chamber is 1,000 mbar. A smooth linear ramp of commanded pressure from 1,000 mbar to 1,250 mbar in 300 mSecs is desired. With unity scaling of the velocity scalar what are the profile generator target velocity and target acceleration values to achieve this profile ramp of the pressure command?

Based on the pressure sensor feedback scaling for this application a command value of -32,767 corresponds to a pressure of 500 mbar, a value of 0 corresponds to a pressure of 1,000 mbar, and a value of +32,767 to a pressure of 1,500 mbar. Therefore a target pressure P of 1,250 mbar equals a command value of V = (P - 1,000) * 32,767 / 500 = 16,384. With the default unity scaling of the velocity scalar (65,536) this results in a target velocity command of the same value, 16,384 counts/cycle time. Because the scaling of the target velocity value is counts/cycle/2^{16} this gives a velocity value to program of 1,073,741,824.

A ramp time of 300 mSecs with the cycle time at the default value of 102.4 µSecs is a time duration of 300 mSecs / 102.4 µSecs/cycle = 2,939 cycles. The target acceleration value is therefore 16,384 counts/cycle / 2,939 cycles = 5.575 counts/cycle^2. Because the scaling of the target acceleration value is counts/cycle^2 / 2^{24}, after multiplying by 2^{24} this gives an acceleration value to program of 93,527,698.

For more information on the scaling used in this application example refer to Section 14.1.2.2, "AnalogCmd Signal With Position/Outer Loop."

The velocity scalar is only used with the position/outer loop control module when an outer loop control function is being performed. When executing as a position loop controller the velocity scalar is not used.

3.3.3 Output Scalar

The position/outer loop provides a general purpose loop output scalar, called Kout, to optimize the effective dynamic range of the control loop. This scalar is most commonly used to increase (or decrease) the sensitivity of the servo gain values Kp, Ki, and Kd. For example if a particular application has a Kp setting of 5, reducing Kout by a factor of ten will result in a Kp setting of 50 having an equivalent effective gain.

Kout is a signed 16 bit number with 1/32,768 scaling, meaning that a scale factor of 1.0 (unity scaling) is expressed with a value of 32,767. Note that the Kout used in the position/outer loop is a different variable, and has a different scaling, then the Kout used with the velocity loop. See Section 4.3, "Settable Parameters" for information on velocity loop module settable variables.

The output value of the position/outer loop is a signed 32 bit quantity with a numerical range of -2,147,483,648 to +2,147,483,647. When the output of the outer/position loop is input to the downstream velocity loop the full 32 bit

output value is utilized. If, however, the position/outer loop output is input to the current loop the output value is divided by 65,536.

3.3.4 Output Limits

The position/outer loop provides programmable polarity-specific output limits, called Outposlimit and Outneglimit. The positive output limit is applied when the loop output value is positive and limits the loop output value to the specified positive output value. Similarly, the negative limit is applied when the output command is negative. Output limiting is a useful safety feature for insuring that the physical or electrical limitations of the actual system are not exceeded by the velocity or torque command sent to the downstream modules.

For both the positive and negative limits zero is an allowed limit value, meaning the user can force the position/outer loop to operate with only one polarity. This may occur in particular with outer loop control functions. For example a circulating pump being used to control temperature may only be able to move in a forward velocity direction. This can be accommodated via a negative output limit of zero.

Both the positive and negative specified output limits are 32 bit numbers. The positive limit range is 0 to 2,147,483,647 while the negative limit range is -2,147,483,648 to 0. The default values are 2,147,483,647 and -2,147,483,648 respectively meaning no output limiting.

3.3.5 Outer Loop Period

The position/outer loop control module allows its loop rate to be separately specified by the user via a parameter known as the outer loop period or Pouter. This outer loop period also sets the update rate of the profile generator should it be specified as the position/outer loop command source.

The most common use of this feature is when the loop is operating as an outer loop controller. Many pressure sensors or other physical sensors provide updates at relatively low rates compared to the Juno cycle time. The value programmed into the outer loop period register represents the number of Juno cycle times that will comprise each position/outer loop cycle.

For example if the Juno cycle time is at the default value of 102.4 µSeconds, a value of 2,000 programmed into the outer loop period register will result in the position/outer loop operating with a cycle time of 102.4 µSec x 2,000 equals 20.48 mSecs (48.8 Hz).

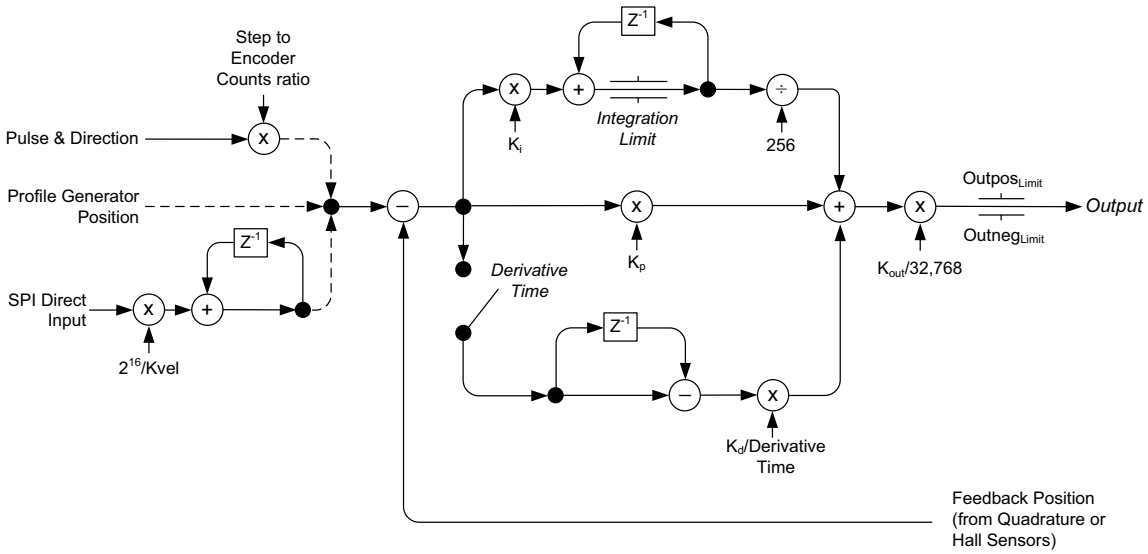
For more information on the master Juno cycle time refer to Section 2.6, "Juno Cycle Time & Loop Rates."

3.3.6 Biquad Filtering

The outer loop controller supports filtering, via Biquad 2, of the *AnalogCmd* input signal should that be selected as the outer loop command source. Biquad filtering may be useful for reducing signal noise or for achieving other optimizations of the outer loop performance. For more information on Juno biquad filtering refer to Section 4.5, "Biquad Filtering."

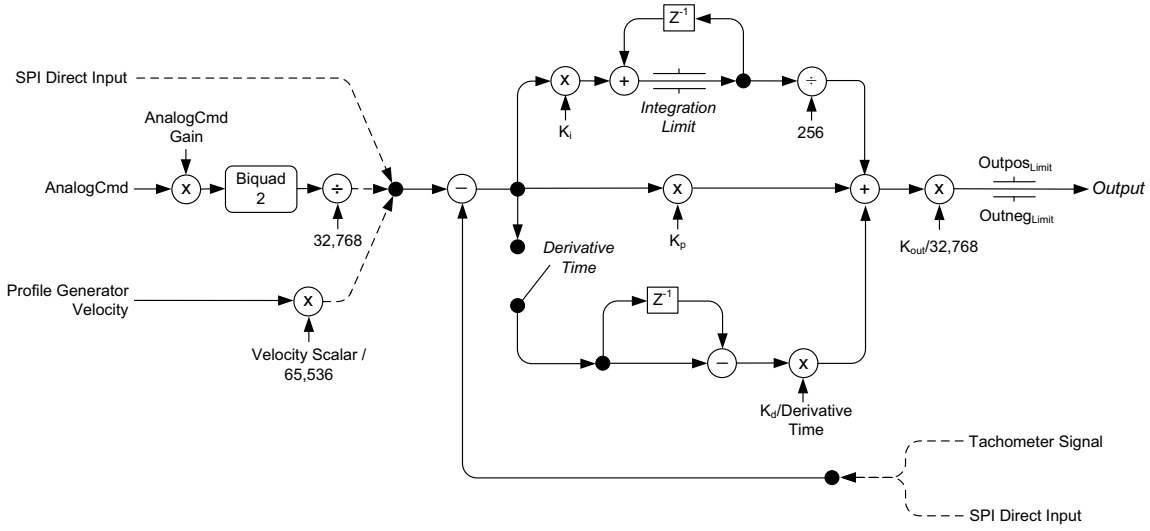
3.4 Position/Outer Loop Calculations

Figure 3-2: Position Loop Calculation Flow



The calculation flow for the position loop function and the outer loop function are somewhat different. The exact scaling and control flow for these loops are provided in Figure 3-2 and Figure 3-3.

Figure 3-3: Outer Loop Calculation Flow



3.5 Motion Error Detection

Under certain circumstances the actual motor position may differ from the commanded position by an excessive amount. Such an excessive position error can indicate a dangerous condition such as motor or encoder failure or excessive mechanical friction.

Juno provides a facility to automatically detect such a condition. A programmable error limit is specified by the user and if the actual error exceeds this programmed threshold a motion error occurs resulting in the corresponding flag

within the Event Status register being set. For more information on this register see Section 8.2.1, "Event Status Register."

To set the motion error limit the **SetLoop** command is used, and to retrieve this programmed limit the **GetLoop** command is used.

The motion error feature operates in the same way whether the loop is being used to control position or an outer loop quantity such as pressure. In each case the programmed limit is numerically compared against the calculated loop error and if exceeded a motion error occurs.

Juno can be programmed to take various actions when a motion error occurs such as bringing the motor to a smooth stop, an abrupt stop, or entirely disabling motor output. The mechanism to program and process these functions is called event handling and is described in detail in Section 8.3, "Event Action Processing."

3.6 Watchdog Timer

Juno provides a facility for detecting when external commands which arrive on a regular basis unexpectedly stop. The ability to detect this condition, known as a watchdog timer, is useful for safely shutting down an axis.

The watchdog feature functions with the profile generator and the SPI direct input command sources. It does not function when *AnalogCmd* or pulse & direction are selected as the command source.

In each case the user selects a watchdog countdown time via the **SetDriveFaultParameter** command in units of cycles. This value can be read back via the **GetDriveFaultParameter** command. The default value for the watchdog countdown timer is 0 which indicates no watchdog function is active.

If a lack of command activity occurs for more than the watchdog countdown period a watchdog error occurs, resulting in the drive exception flag of the Event Status register being set. For more information see Section 8.2.1, "Event Status Register."

Juno can be programmed to take various actions when a watchdog timeout occurs such as bringing the motor to a smooth stop, an abrupt stop, or entirely disabling the motor output. The mechanism to program and process these functions is called event handling and described in detail in Section 8.3, "Event Action Processing."

3.7 Position/Outer Loop Operation Startup

Initializing a cascaded outer loop/velocity loop control function such as that shown in Figure 2-17 requires special attention to the startup procedure. In general it is recommended that the velocity loop be enabled and used to stably operate the system before enabling the position/outer loop.

In addition, if the *AnalogCmd* or SPI direct input command sources are used, at the time the position/outer loop is enabled the user should insure that the command word closely matches the measured outer loop value so that there is no discontinuity.

If the profile generator is selected as the command source, at the time the position/outer loop is enabled the profile generator's target and commanded velocity will automatically be set to the measured outer loop value. In this way there should be no discontinuity of command at the time the position/outer loop is enabled.

To enable or disable Juno control modules the **SetOperatingMode** command is used.

3.8 Enabling and Disabling the Position/ Outer Loop Module

If disabled, no calculations occur in this module and no output value is available to downstream control loop modules. In addition, any accumulating registers such as the PID integral sum will be set to zero.

If enabled, the user specified command sources and feedback sources will be applied to this module and will no longer apply to downstream enabled modules. In addition, calculations will immediately begin and the calculated loop output value will be used by the next enabled downstream module. At the time this module is enabled the integral sum is initialized so that the loop output does not change abruptly.

To disable the position/outer loop module the command **SetOperatingMode** is used. The value set using this command can be read using **GetOperatingMode**.

A previously disabled position/outer loop module may be re-enabled in a number of ways. If output was disabled using the **SetOperatingMode** command, then another **SetOperatingMode** command may be issued. If disabled as part of an automatic safety event-related action (see Section 8.3, "Event Action Processing" for more information), then the command **RestoreOperatingMode** is used.

The default condition of the position/outer loop module is disabled, therefore if use of this module is desired the external controller must send a **SetOperatingMode** command to enable the module.

4. Velocity Loop

4

In This Chapter

- Selecting the Command Source
- Selecting the Feedback Source
- Settable Parameters
- Velocity Loop Calculations
- Biquad Filtering
- Motion Error Detection
- Watchdog Timer
- Enabling and Disabling the Velocity Loop

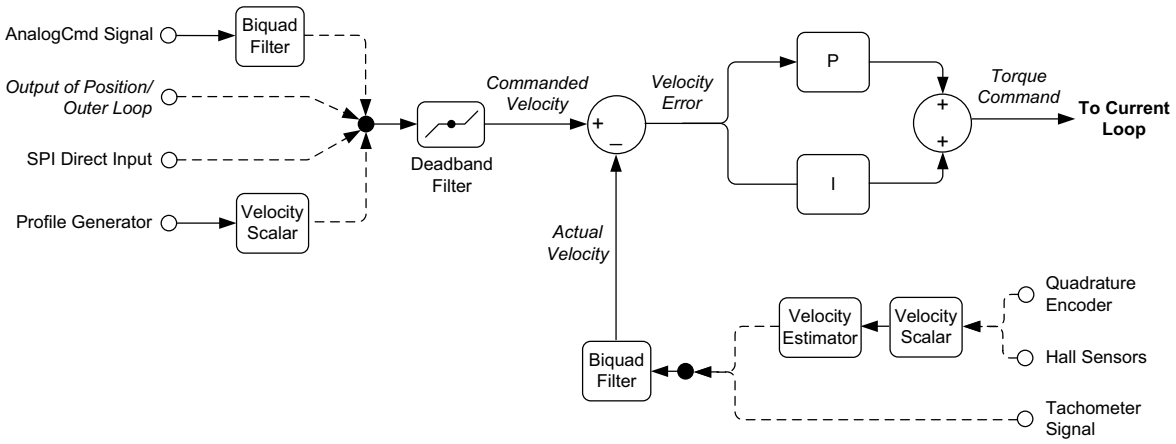


Figure 4-1: Velocity Loop Control Flow

Figure 4-1 provides a summary of the control flow of Juno's velocity loop control module. This module provides a programmable velocity loop function supporting a wide range of applications.

The velocity loop module is used whenever a high performance velocity control function is desired. Although most often this module is programmed to directly input commands from external circuitry, it is also possible for the velocity command to come from the profile generator or from the upstream position/outer loop module.

This loop module utilizes the desired velocity command and a measured velocity to develop a loop error value which is passed through a PI (proportional, integral) filter and output to the next enabled downstream control module, usually the current loop module.

The velocity loop also provides sophisticated filtering capability in the form of biquad filters located in the velocity feedback path and in the analog command input path, as well as a deadband filter at the velocity command output. These filters can be used to smooth velocity and improve stability in the motion system being controlled.

4.1 Selecting the Command Source

The velocity loop allows several command sources to be selected. Most common are either the direct input SPI port or the ***AnalogCmd*** signal. Both the direct input SPI command and the analog command input value are signed 16 bit

quantities representing the commanded velocity. Alternatively, the profile generator can be selected as the velocity command source. In this case the profile generator's 32 bit velocity register value will be used after being scaled by a user-specified velocity scalar value. Finally, if the position/outer loop control module is enabled the output of that module will be the command source for the velocity loop module.

To select the command source the **SetDriveCommandMode** command is used. To read this value back the **GetDriveCommandMode** command is used. To read the current value of the direct input SPI register the command **GetLoopValue** is used. To read the current value of the *AnalogCmd* signal register the command **GetLoopValue** is used. To read the current value of the velocity register the command **GetCommandedVelocity** is used.

4.2 Selecting the Feedback Source

The velocity loop supports three selectable feedback sources; quadrature encoder feedback, Hall sensors, and the *Tachometer* analog input signal. Both the quadrature and Hall inputs create a velocity by successive subtraction. That is, the velocity is calculated from the difference between the current position and the previous position.

To select the feedback source to encoder or Hall sensors the **SetLoop** command is used with a setting of encoder, and then **SetEncoderSource** command is used with a setting of either quadrature encoder or Halls. To select the *Tachometer* signal the **SetLoop** command is used with a setting of tachometer. To read these values back the **GetEncoderSource** or **GetLoop** commands are used. To read the 16-bit value of the *Tachometer* input signal register the command **GetLoopValue** is used. To read the 32-bit value of the feedback position (whether Halls or quadrature encoder are selected as the source) the **GetActualPosition** command is used.

4.3 Settable Parameters

To control the velocity loop up to seven parameters are set by the user; Kp, Ki, Ilimit, Kvel, Kout, DBlow, and DBhigh. Two of these are gain factors for the PI (proportional, integral) controller, one is a limit for the integral contribution, one is a scale factor that is applied only when the command source is set to profile generator or when the feedback source is quadrature encoder or Halls, one is an output scale factor that is always applied, and two are the lower and upper deadband filter limits.

Term	Name	Default Value	Representation & Range
Kp	Proportional gain	0	Unsigned 16 bits (0 to 32,767)
Ki	Integral gain	0	Unsigned 16 bits (0 to 32,767)
Ilimit	Integration limit	0	Unsigned 16 bits (0 to 32,767)
Kvel	Velocity scalar	65,536	Unsigned 32 bits (0 to 2,147,483,647)
Kout	Output scalar	256	Signed 16 bits (-32,768 to +32,767)
DBlow	Deadband lower limit	0	unsigned 32 bits (0 to 2,147,483,647)
DBhigh	Deadband upper limit	0	unsigned 32 bits (0 to 2,147,483,647)

All of these parameters are set with the **SetLoop** command and read back with the **GetLoop** command.

Determining correct parameters for the Kp, Ki, and Ilimit gains can be done in a number of ways. The easiest is to utilize the tuning facilities provided within PMD's Pro-Motion software package.

In addition to these settable parameters there are settable parameters associated with the biquad filter, should it be used. Refer to Section 4.5, "Biquad Filtering" for more information.

4.3.1 Velocity Scalar

The velocity scalar register, called Kvel, is used by several Juno control modules including the velocity loop module. Kvel is an unsigned 32 bit number with 1/65,536 scaling meaning that a scale factor of 1.0 (unity scaling) is expressed with a value of 65,536.

When used with the velocity loop Kvel is most commonly used to scale the encoder or Hall sensor feedback when functioning as the source of the actual velocity measurement and when the ***AnalogCmd*** signal or the direct-input SPI port is used for the velocity command. The following example illustrates this.

Example

An application will utilize the ***AnalogCmd*** signal to command a velocity controller as detailed in the example in Section 7.1, "Settable Parameters." In this application the maximum commandable velocity is 13.825 encoder counts/cycle (where cycle is a Juno cycle time of 102.4 µSecs). To provide a minimum recommended scaling overhead of ~30% we will scale the ***AnalogCmd*** so that the maximum negative and positive voltages command velocities of -20.0 counts/cycle and +20.0 counts/cycle respectively.

From Figure 4-3 the ***AnalogCmd*** value is scaled up by a factor of 32,768. The velocity scalar required to implement the desired scaling is therefore Kvel = 32,767 * 32,768 / 20.0 = 53,685,452. This value of the velocity scalar will result in a velocity command range of +/- 20.0 counts/cycle at the ***AnalogCmd*** signal.

Note that if in this application Juno's default Kvel value of 65,536 had been used the maximum commandable range of velocity would be V = +/- 32,767 * 32,768 / 65,536 = +/- 16,384 counts/cycle. Since the actual application range is only +/- 20 counts/cycle this means that only one tenth of 1% of the analog signal input range would be utilized to control the desired range of velocity. This example illustrates the importance of correctly setting the velocity scalar for applications which use the ***AnalogCmd*** or SPI direct input ports in combination with the encoder or Hall sensors for velocity feedback.

4.3.2 Output Scalar

The velocity loop provides a general purpose loop output scalar, called Kout, to optimize the effective dynamic range of the control loop. Kout should be set so that the maximum expected output value is 50% - 75% of the encodable numerical range.

Kout is a signed 16 bit number with 1/256 scaling, meaning that a scale factor of 1.0 (unity scaling) is expressed with a value of 256. Note that the Kout used in the velocity loop is a different variable, and has a different scaling, then the Kout used with the position/outer loop.

4.3.3 Deadband Filter

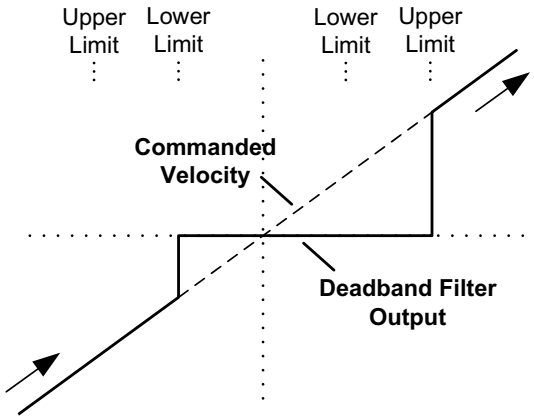


Figure 4-2: Deadband Filter Format

Juno ICs provide a deadband filter associated with the velocity loop that may be useful for reducing hunting when the command velocity is around zero. The deadband filter may be used with any of the velocity loop command sources but is most commonly used when the velocity loop is commanded by the Juno outer loop, or by an outer loop implemented in an external processor, using analog or SPI direct command.

Two parameters are provided to control the deadband filter, the deadband lower limit, denoted as DB_{low} and the deadband upper limit, denoted as DB_{high}. Each of this parameters are unsigned 32-bit quantities with a range of 0 to 2,147,483,647.

At each calcuatlion cycle the deadband filter uses both the commanded velocity (the input to the filter) and the deadband filter output to determine a new deadband filter output value. This is determined as follows:

If the previous deadband filter output value was zero than the absolute value of the commanded velocity is compared to the deadband upper limit. If the limit is exceeded the commanded velocity is used as the new deadband filter output. If the limit is not exceeded a value of zero is used as the new deadband filter output.

If the previous deadband filter output value was non-zero than the absolute value of the commanded velocity is compared to the deadband lower limit. If the limit is exceeded the commanded velocity is used as the new deadband filter output. If the limit is not exceeded a value of zero is used as the new deadband filter output.

Use of two deadband limits, an upper limit and a lower limit allows a hysteresis function to be supported. The lower limit may be equal to the upper limit but may never exceed it. A value of zero in both the upper and lower limits effectively turns off the deadband filter, which is Juno's default condition.

Figure 4-2 shows the form of the deadband filter for a commanded velocity that traverses linearly from a large negative command value, through a zero value, to a large positive value. This graph shows the relationship between the deadband output value as the commanded velocity value approaches and continues past zero. For a commanded value that traverses from a positive to a negative commanded value (effectively reversing the direction of the arrows shown) the form is similar in shape but shifted down and to the left by the difference between upper limit and lower limit value.

4.4 Velocity Loop Calculations

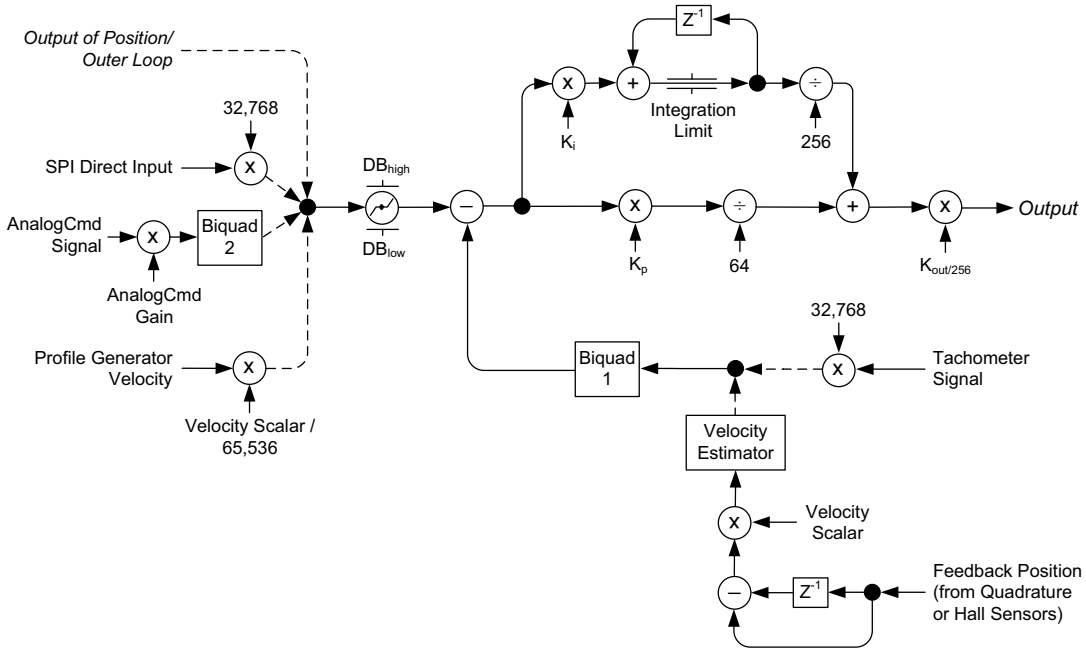


Figure 4-3: Velocity Loop Calculation Flow Diagram

The exact scaling and control flow for the Juno velocity loop is provided in Figure 4-3.

4.5 Biquad Filtering

As shown in Figure 4-3 a biquad filter is included in the feedback pathway of the Juno velocity loop as well as at the *AnalogCmd* signal command source input. For applications that may require it, biquads provide a sophisticated tool for smoothing the functioning of the system, reducing vibration, and reducing resonances in the system being controlled.

Biquads can be programmed to create different filtering functions including lead/lag, band pass, low pass, notch, and other filter types. A typical system development process involves making an open loop frequency scan of the system to be controlled to learn about dead spots and resonances in its frequency response. From this information the biquad filter can be programmed such that these system response problems are reduced or eliminated.

4.5.1 Settable Parameters

There are two biquad filters labelled biquad1 and biquad2. Biquad1 is located in the motor feedback path, while biquad2 is located at the *AnalogCmd* input signal. The following table lists the parameters that are set to control the operation of either of the two biquad filters.

Term	Default Value	Representation & Range
Coefficient A_1	1,533,916,891	signed 32 bits (–2,147,483,648 to +2,147,483,647)
Coefficient A_2	-547,827,461	signed 32 bits (–2,147,483,648 to +2,147,483,647)
Coefficient B_0	+87,652,394	signed 32 bits (–2,147,483,648 to +2,147,483,647)
Coefficient B_1	0	signed 32 bits (–2,147,483,648 to +2,147,483,647)
Coefficient B_2	0	signed 32 bits (–2,147,483,648 to +2,147,483,647)
Biquad1 Enable	Enabled	Single bit field with value of enabled or disabled.
Biquad2 Enable	Disabled	Single bit field with value of enabled or disabled.

To set any of these parameters the command **SetLoop** is used. To read back these parameters, the command **GetLoop** is used.

The default values for both biquad1 and biquad2 provide a critically damped low pass filter with a cutoff at a frequency of approximately 675 Hz.

For more information on setting the Juno velocity loop biquad filters to achieve specific filtering functions please refer to the *Juno Velocity & Torque Control IC Programming Reference.*

Before changing biquad filters, the entire biquad should be disabled. This is accomplished using the **SetLoop** command. Once the entire set of new biquad parameters have been loaded in, the biquad can then be safely re-encoded using the **SetLoop** command.

4.5.2 Biquad Filter Calculations

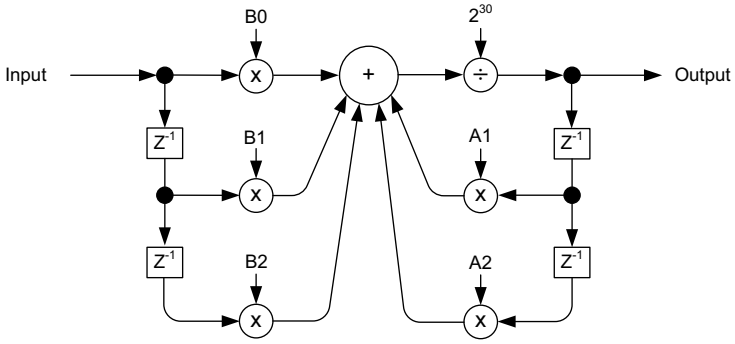


Figure 4-4: Biquad Calculation Flow

The exact scaling and control flow for the two Juno biquad filters is shown in Figure 4-4.

4.6 Motion Error Detection

Under certain circumstances the actual motor velocity may differ from the commanded velocity by an excessive amount. Such an excessive velocity error often indicates a potentially dangerous condition such as motor or encoder failure, excessive load, or high mechanical friction.

Juno ICs provide a facility to automatically detect such a condition. A programmable error limit is specified by the user and if the actual velocity error limit exceeds this programmed threshold a motion error occurs, resulting in the corresponding flag within the event status register being set. For more information see Section 8.2.1, "Event Status Register."

To set the motion error limit the **SetLoop** command is used, and to retrieve this programmed limit the **GetLoop** command is used.

Juno can be programmed to take various actions when a motion error occurs such as bringing the motor to a smooth stop, an abrupt stop, or entirely disabling the motor output. The mechanism to program and process these functions is called event handling and described in detail in Section 8.3, "Event Action Processing."

If both the velocity loop and the position/outer loop control modules are active the motion error checking function will occur within the position/outer loop, and no velocity error checking will occur.

4.7 Watchdog Timer

Juno provides a facility for detecting when external commands which arrive on a regular basis unexpectedly stop. The ability to detect this condition, known as a watchdog timer, is useful for safely shutting down an axis.

For a detailed description of this feature see Section 3.6, "Watchdog Timer."

4.8 Enabling and Disabling the Velocity Loop

If disabled, the output from the position/outer loop module, if enabled, will pass directly to the current loop module with no velocity control being performed.

If enabled, the user specified command sources and feedback sources will be applied to this module and will no longer apply to downstream enabled modules. In addition, calculations will immediately begin and the calculated loop output value will be used by the next enabled downstream module.

To disable the velocity loop module the command **SetOperatingMode** is used. The value set using this command can be read using **GetOperatingMode**.

A previously disabled velocity loop module may be re-enabled in a number of ways. If output was disabled using the **SetOperatingMode** command, then another **SetOperatingMode** command may be issued. If disabled as part of an automatic safety event-related action (see Section 8.3, "Event Action Processing" for more information), then the command **RestoreOperatingMode** is used.

The default condition of the velocity loop module is disabled, therefore to enable the velocity loop, the external controller must send a **SetOperatingMode** command enabling the module.

To read the instantaneous actual state of the operating mode, the command **GetActiveOperatingMode** is used.

This page intentionally left blank.

5. Current Loop

5

In This Chapter

- Selecting the Command Source
- Settable Parameters
- Current Loop Operation
- Watchdog Timer
- Enabling and Disabling the Current Loop Module

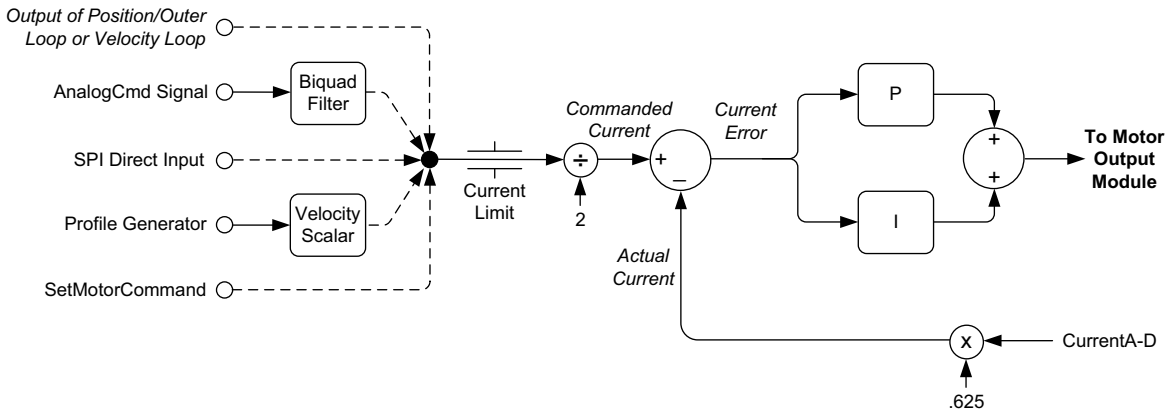


Figure 5-1: Current Loop Control Flow Diagram

Figure 5-1 provides a summary of the control flow of Juno's current loop control module. Current control is a technique used for DC Brush, Brushless DC, and step motors for controlling the current (and therefore the torque) through each winding of the motor. By controlling the current, response times improve, motor efficiency is higher, and motion smoothness increases.

The Juno digital current loop utilizes the desired current for each motor winding along with the actual measured current which is input by direct analog signal input into the Juno IC. The desired current and measured current are then subtracted to develop a current error, which is passed through a PI (proportional, integral) filter to generate an output voltage command for each motor coil. The output command for each coil is then passed to the motor output module which generates the precise PWM (pulse width modulation) timing output signals to control external switching circuitry.

For DC Brush motors, which are single phase devices, the current loop does not require information about the motor's rotor angle. For Brushless DC and step motors however the motor's rotor angle is required. For information about Brushless DC commutation see Chapter 9, *Brushless DC Motor Control*. For information about step motor microstepping waveform generation see Chapter 10, *Step Motor Control*.

5.1 Selecting the Command Source

The current loop allows several command sources to be selected. If either the velocity loop or the position/outer loop is enabled the output of those modules will be the command source for the current loop module.

If no upstream modules are enabled either the direct input SPI port or the *AnalogCmd* signal can be used to specify a command from external circuitry. Both the direct input SPI command and the analog command input value are signed 16 bit quantities representing the commanded current (torque) value.

Alternatively, the profile generator can be selected as the current loop command source. This may be useful if a ramped current command is desired. The profile generator's 32 bit velocity register value will be used after being scaled by a user-specified velocity scalar value.

Finally, the current loop supports an additional command source consisting of a signed 16-bit register set via the **SetMotorCommand** command. This direct set user command is sometimes useful for testing or calibration during system setup. To select this register as the command source the **SetOperatingMode** command is used.

To select the command source the **SetDriveCommandMode** command is used. To read this value back the **GetDriveCommandMode** command is used. To read the current value of the direct input SPI register the command **GetFOCValue** is used. To read the current value of the *AnalogCmd* signal register the command **GetFOCValue** is used. To read the current value of the velocity register the command **GetCommandedVelocity** is used. To read the value of the **SetMotorCommand** the **GetMotorCommand** is used.

5.2 Settable Parameters

To control the current loop up to five parameters are specified by the user; Kp, Ki, Ilimit, Kvel, and Currentlimit. Two of these are gain factors for the PI (proportional, integral) controller, one is a limit for the integral contribution, one is a scale factor used only when the command source is profile generator,and one is a limit to the incoming current command. These five parameters have the following ranges and formats:

Term	Name	Default Value	Representation & Range
$Kp_{current}$	Current loop proportional gain	0	Unsigned 16 bits (0 to 32,767)
$Ki_{current}$	Current loop integrational gain	0	Unsigned 16 bits (0 to 32,767)
$Ilimit_{current}$	Current loop integration limit	0	Unsigned 16 bits (0 to 32,767)
K_{vel}	Velocity scalar	65,536	Unsigned 32 bits (1 to 2,147,483,647)
Currentlimit	Current limit	32,767	Unsigned 16 bits (0 to 32,767)

To set the gain parameter and the integration limit the command **SetFOC** is used. To read back these parameters the command **GetFOC** is used. To set Kvel, the **SetLoop** command is used and **GetLoop** is used to read the value back. To set the current limit the command **SetCurrentLimit** is used and **GetCurrentLimit** is used to read this value back. Note that for Brushless DC and step motors the gain values for the phase A and B loops can be set independently while for single-phase DC brush motors only the phase A loop parameters are used.

Determining correct Kp, Ki, and Ilimit parameters for the current loop controller gains can be done in a number of ways. The easiest is to utilize the auto-tuning facility provided within PMD's Pro-Motion software package. Parameters derived using this procedure may or may not be optimized for your system but will be adequate for most applications and a good starting point.

Please note that it is the responsibility of the user to determine the suitability of all control parameter values, including those determined by auto-tuning, for use in a given application.

5.2.1 Velocity Scalar

The velocity scalar register is used in connection with the current loop when the command source is set to profile generation. For more information on this register refer to Section 4.3.1, "Velocity Scalar."

5.2.2 Current Limit

Juno provides a settable limit to the magnitude of the commanded current via the register called Currentlimit. The current limit functions by capping the magnitude of the commanded current to the specified value. For example if the specified current limit is 10,000 an incoming command of +12,345 would be set to +10,000 and an incoming command of -12,345 would be set to -10,000.

Limiting the commanded current is a useful safety feature for insuring that the physical or electrical limitations of the actual system are not exceeded.

Current limit is an unsigned 16 bit number with a range 0 - to 32,767. The default value is 32,767 meaning no limiting is applied.

> The current limit only limits the commanded current. Whether or not the motor current is actually limited to this threshold is a function of whether or not the current loop is functioning properly.

5.2.3 Biquad Filtering

The current loop supports filtering, via Biquad 2, of the *AnalogCmd* input signal. This may be useful for reducing signal noise or achieving other optimizations of the current loop performance. For more information on Juno biquad filtering see Section 4.5, "Biquad Filtering."

5.3 Current Loop Operation

For DC Brush and step motors the current loop method is fixed and does not need to be set to be used. For Brushless DC motors users may specify one of two current control methods. The large majority of applications will use field oriented control (FOC) for Brushless DC motor control. FOC usually provides the highest top speeds and more energy efficient operation of the motor.

Third leg floating is an option that may be considered with Hall-commutated Brushless DC motors. In that configuration third leg floating can sometimes provide a higher top speed than FOC. Compared to FOC, third leg floating drives only two of three legs at any instant with the third, non-driven leg, floating.

For Brushless DC motors to select which type of control method will be used, use the command **SetCurrentControlMode**. To read the value set using this command, use **GetCurrentControlMode**. When third leg floating current control is selected the commutation mode set via the command **GetCommutationMode** must be set to Hall-based. For FOC current control the commutation mode can be set to either Hall-based or encoder-based. For more information on Brushless DC motor commutation see Chapter 9, *Brushless DC Motor Control*

5.4 Watchdog Timer

Juno provides a facility for detecting when external commands which arrive on a regular basis unexpectedly stop. The ability to detect this condition, known as a watchdog timer, is useful for safely shutting down an axis.

For a detailed description of this feature see Section 3.6, "Watchdog Timer."

5.5 Enabling and Disabling the Current Loop Module

If disabled, output will pass directly to the power stage module with no current control being performed. The most common use of this is to run the amplifier in voltage mode, which may be useful under some conditions for calibration or testing.

If enabled, the user specified command sources and feedback sources will be applied to this module and will no longer apply to downstream enabled modules. In addition, calculations will immediately begin and the calculated loop output value will be used by the motor output module.

To disable the current loop module the command **SetOperatingMode** is used. The value set using this command can be read using **GetOperatingMode**.

A previously disabled current loop module may be re-enabled in a number of ways. If output was disabled using the **SetOperatingMode** command, then another **SetOperatingMode** command may be issued. If disabled as part of an automatic safety event-related action (see Section 8.3, "Event Action Processing" for more information), then the command **RestoreOperatingMode** is used.

The default condition of the current loop module is disabled, therefore to begin motor operations, the external controller must send a **SetOperatingMode** command enabling the current loop module.

To read the instantaneous actual state of the operating mode, the command **GetActiveOperatingMode** is used.

6. Motor Output

In This Chapter

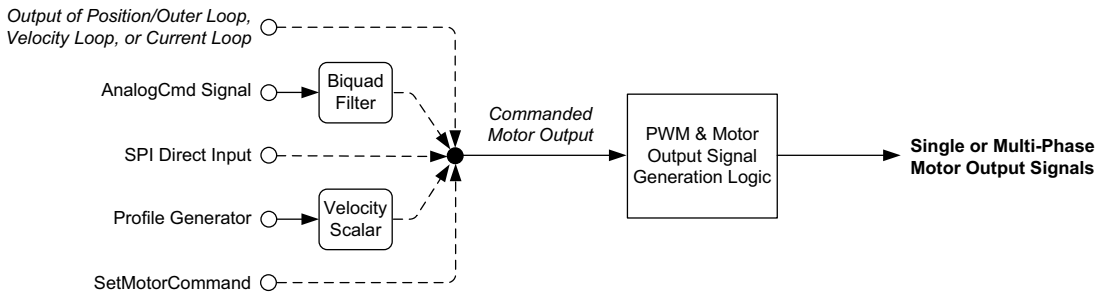


Figure 6-1: Motor Output Control Flow

The purpose of Juno's motor output module is to generate PWM (Pulse Width Modulation) and related motor output signals for use by external switching amplifier circuitry.

Juno ICs provide two different motor output methods, known as PWM High/Low, and Sign/Magnitude PWM. The switching control mode that is used when Juno's current control facility is utilized is PWM High/Low mode. Sign/Magnitude PWM is typically used with single-IC amplifiers and bridges that themselves perform current control, or that directly input sign and PWM magnitude control signals.

For a detailed description of the signals associated with these functions refer to the *MC78113 Electrical Specifications.*

6.1 Selecting the Command Source

The Motor Output Module allows several command sources to be selected. Most common by far is that one of the 'upstream' control loops is enabled in which case these modules provide the motor output command. These upstream loops are the position/outer loop, the velocity loop, and the current loop.

If no upstream modules are enabled Juno operates in voltage mode (no current loop active) and either the direct input SPI port or the ***AnalogCmd*** signal can be used to specify a voltage command via external circuitry. Both the direct input SPI command and the analog command input value are signed 16 bit quantities representing the desired voltage.

Alternatively, the profile generator can be selected as the motor output command source. This may be useful if a ramped voltage is desired. The profile generator's 32 bit velocity register value will be used after being scaled by a user-specified velocity scalar value.

Finally, the motor output module supports an additional command source consisting of a signed 16-bit register set via the **SetMotorCommand** command. This direct set user command is sometimes useful for testing or calibration during system setup. To select this register as the command source the **SetOperatingMode** command is used.

To select the command source the **SetDriveCommandMode** command is used. To read this value back the **GetDriveCommandMode** command is used. To read the current value of the direct input SPI register the command **GetLoopValue** is used. To read the current value of the *AnalogCmd* signal register the command **GetLoopValue** is used. To read the current value of the velocity register the command **GetCommandedVelocity** used. To read the value of the **SetMotorCommand** the **GetMotorCommand** is used.

6.2 PWM High/Low Motor Output Mode

Juno ICs can control high-efficiency MOSFET or IGBT power stages with individual high/low switch input control. A different configuration is used for each motor type:

- DC Brush motors are driven in an H-Bridge configuration consisting of 4 switches.
- Brushless DC motors are driven in a triple half-bridge configuration consisting of 6 switches.
- Step motors are driven in a two H-Bridge configuration consisting of 8 switches.

In PWM High/Low mode each signal carries a variable duty cycle PWM signal. A zero desired motor command results in the high side and low side being active for the same amount of time. Positive motor commands are encoded as a high-side duty cycle greater than 50%, and negative motor commands are encoded as a duty cycle less than 50%. This is shown in Figure 6-2.

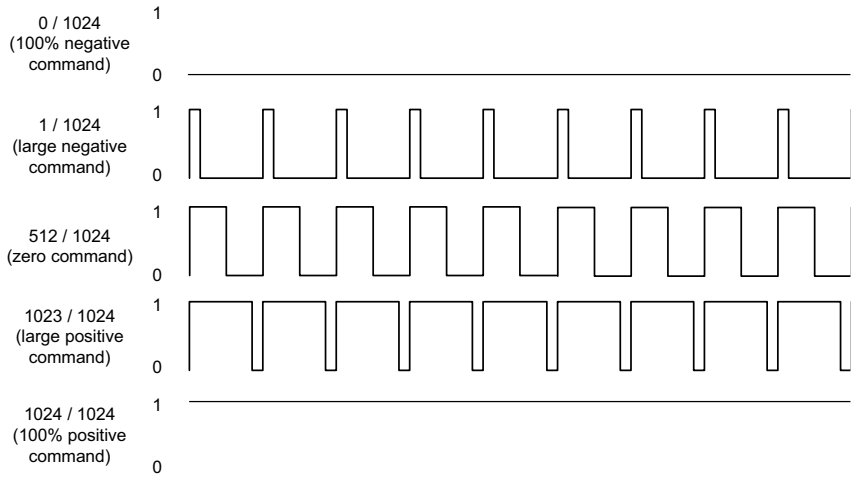


Figure 6-2: PWM High/Low Encoding

In PWM High/Low mode two output pins are used per motor terminal, allowing separate high-side/low-side control of each bridge switch. In this scheme the high side output and the low side output are never active at the same time, and there is generally a period of time when neither output is active. This period of time is called the dead time, and provides a shoot through protection function for MOSFET or IGBT switches.

In addition to dead time, some high side switch drive circuitry requires a minimum amount of off time to allow the charge pump circuitry to refresh. This parameter specifies this refresh time and has units of nSecs. The related parameter of refresh time period, which is the time interval between these off time refreshes has units of current loop cycles.

It is also possible to control the maximum allowed PWM duty cycle. This may be useful to limit the effective voltage presented to the motor windings, or to provide some other needed off-time for the switching amplifier circuitry. This PWM Limit parameter along with all of these PWM control parameters is set using the **SetDrivePWM** command.

6.2.1 Settable Parameters

The following table shows the settable control parameters when PWM High/Low motor output mode is used:

Parameter	Range & Units	Description
PWM Switching Frequency	20 kHz 40 kHz 80 kHz 120 kHz	Higher inductance motors should be set for 20 kHz. Lower inductance motors may use 40, 80, or 120 kHz to maximize current control accuracy and minimize heat generation. The default value for this parameter is 20 kHz.
PWM Dead Time	0-32,767 nSec	The correct setting of this parameter depends on the specific switching circuitry used. See the manufacturer's data sheet for more information. The default value for this parameter is 16,879 nSec.
PWM Refresh Time	0-32,767 nSec	Some high-side switch drive circuitry requires a minimum amount of off time, applied at a programmable period interval, to allow the charge pump circuitry to refresh. The default value for this parameter is 32,767 nSec.
PWM Refresh Period	1-32,767 cycles	Some high-side switch drive circuitry requires a minimum amount of off time, applied at a programmable period interval, to allow the charge pump circuitry to refresh. The default value for this parameter is 1 cycle.
PWM Limit	0-16,384 % output/163.84	This parameter allows the maximum PWM duty cycle to be set. The default value for this parameter is 16,384 which corresponds to 100% output.
PWM Signal Sense	16-bit mask	This parameter allows the signal sense of the PWM output signals to be specified. A 1 in the bit mask indicates active high, a 0 indicates active low. The default value is all signals active high.

All of the above parameters are set using the **SetDrivePWM** command and are read back using the **GetDrivePWM** command.

For much more information on the function of the motor output control settings and signal outputs refer to the *MC78113 Electrical Specifications*.

6.3 Sign/Magnitude PWM Output Mode

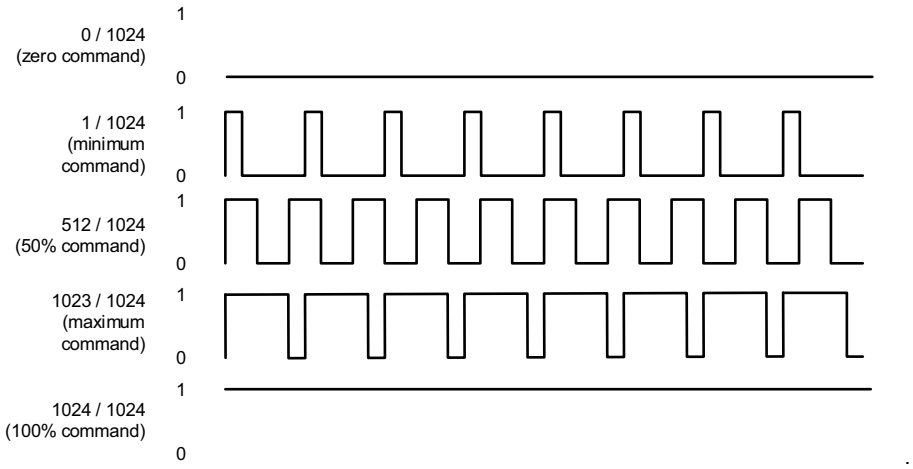


Figure 6-3: Sign/ Magnitude PWM Encoding

In Sign/Magnitude PWM mode, two pins are used to output the motor command information for each motor phase. One pin carries the PWM magnitude, which ranges from 0 to 100% as shown in Figure 6-3. A high signal on this pin means the motor coil should be driven with voltage. A second pin outputs the sign of the motor command by going high for positive sign, and low for negative.

In Sign/Magnitude PWM control mode only DC Brush and step motors can be controlled. Brushless DC motors can not be controlled in this mode.

6.3.1 Settable Parameters

The following table shows the settable parameters when Sign/Magnitude PWM Control motor output mode is used.

Parameter	Range & Units	Description
PWM Switching Frequency	20 kHz 40 kHz 80 kHz 120 kHz	Higher inductance motors should be set for 20 kHz. Lower inductance motors may use 40, 80, or 120 kHz to maximize current control accuracy and minimize heat generation. The default value for this parameter is 20 kHz.
PWM Limit	0-16,384 % output/163.84	This parameter allows the maximum PWM duty cycle to be set. The default value for this parameter is 16,384 which corresponds to 100% output.
PWM Signal Sense	16-bit mask	This parameter allows the signal sense of the PWM output signals to be specified. A 1 in the bit mask indicates active high, a 0 indicates active low. The default value is all signals active high.

All of the above parameters are set using the **SetDrivePWM** command and are read back using the **GetDrivePWM** command.

For much more information on the function of the motor output control settings and output signals refer to the *MC78113 Electrical Specifications*.

6.4 AmplifierEnable Signal

Regardless of whether the motor output mode is set to PWM High/Low or Sign/Magnitude PWM, Juno provides an *AmplifierEnable* signal output that indicates whether the external amplifier circuitry is active or not. While not all external amplifiers will require or provide such an input control, this signal is useful for general safety purposes, as well as to simplify the task of ensuring startup without jogging the motor after powerup.

The output of this signal corresponds directly to the state of the motor output bit and the braking bit of the active operating mode register. If either of these bits are on, the *AmplifierEnable* signal is active. If both bits are off, this signal is inactive.

For additional information on this signal refer to the *MC78113 Electrical Specifications.*

6.5 Brake Signal

Juno's *Brake* signal input provides a high speed PWM output override function that may be useful for safety protection when using Brushless DC or DC Brush motors. When this input is active PWM output is driven to one of two user programmable states; a braking state or a fully disabled state.

When a brake command is asserted with the braking function programmed Juno controls the PWM switches in such a way that current will circulate within the coils and generate resistive back-EMF torque resulting in a deceleration of the motor. When a brake command is asserted with the fully disabled mode programmed no back-EMF braking is applied to the motor. The PWM switches will be disabled and the motor will "free wheel" and decelerate more slowly to a stop.

The actual motor motion response after a brake signal is applied is application dependent. Particularly in applications where external forces exist on the motor, after the brake signal is applied the motor may not decelerate as described above or may even accelerate. It is the responsibility of the user to determine whether, and how, the *Brake* signal should be used in their application.

To program this function the **SetEventAction** command is used. If a brake function occurs, to re-enable normal output the **ResetEventStatus** command along with the **RestoreOperatingMode** command is used. For more information on Juno event processing see Section 8.3, "Event Action Processing."

6.6 Enabling and Disabling the Motor Output Module

If disabled, a "zero" command will be sent to the motor, or to each phase of the motor for multi-phase motors such as brushless DC or microstepping motor. Depending on the motor output signal format, this zero command will be represented in different ways. In particular, for PWM High/Low mode both the high side and low side switches will be commanded off. For Sign/Magnitude PWM, both the sign signal and the magnitude signal will be driven to their deactive state as set with the **SetSignalSense** command.

Note that disabling the motor output module may or may not immediately stop the motor. Disabling this module has the effect of "free-wheeling" the motor, which means the motor may stop, coast, or even accelerate if a constant external force exists such as the force of gravity on a vertical axis.

To disable the motor output module the command **SetOperatingMode** is used. The value set using this command can be read using **GetOperatingMode**.

A disabled motor output module may be re-enabled in a number of ways. If the module was disabled using the **SetOperatingMode** command, then another **SetOperatingMode** command may be issued. If this module was disabled as part of an automatic event-related action (see Section 8.3, "Event Action Processing," for more information), then the command **RestoreOperatingMode** is used.

The default condition of the motor output module is disabled, therefore to begin motor operations, the external controller must send a **SetOperatingMode** command enabling the motor output module.

7. Internal Profile Generation

7

In This Chapter

- Settable Parameters
- Programmed Stops
- Enabling and Disabling the Profile Generator Module
- Profile Generator as Loop Command Source

Juno ICs include an internal profile generator that allows arbitrary contours of the commanded position, outer loop quantity, velocity, current, or voltage to be generated. The profile generator is used in conjunction with host commands to specify one or more move profiles.

To control the profile generator the user specifies a desired target acceleration, deceleration, and velocity. Using these target parameters Juno's profile generator performs calculations to determine the instantaneous position, velocity, and acceleration of the profile at any given moment. These instantaneous profile values are called the commanded values. During profile execution, some or all of the commanded values will continuously change as the profile is generated.

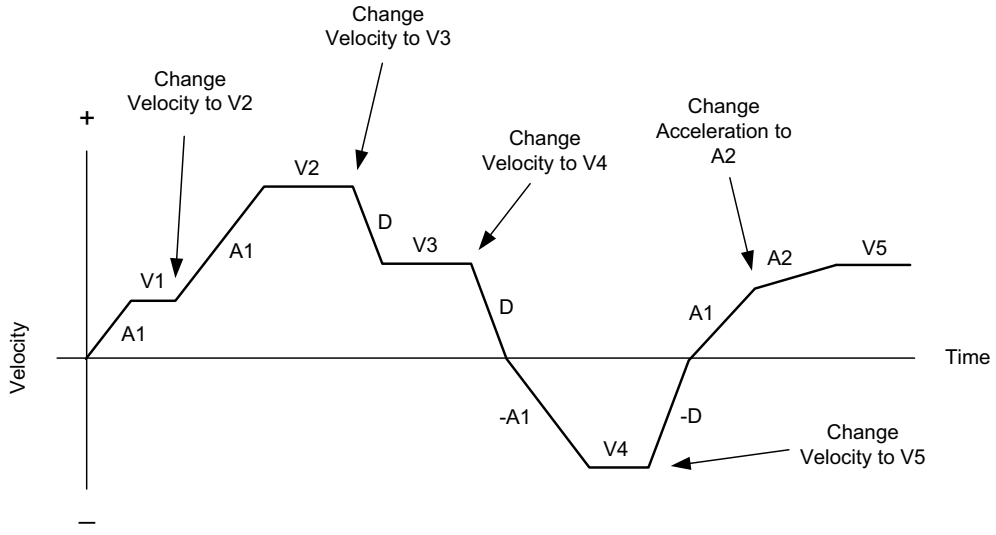


Figure 7-1: Internally Generated Velocity Profile

The profile is executed by continuously accelerating at the user-specified target acceleration rate until the user-specified target velocity is reached. The sign of the velocity parameter determines the initial direction of motion. Therefore the velocity value sent to Juno can have a positive value (for positive direction motion), or a negative value (for negative direction motion).

The axis decelerates at the user-specified target deceleration when a new velocity is specified with a smaller value (in magnitude) than the present velocity, or when a new velocity has a sign that is opposite to the present direction of travel. When a decelerating axis decelerates through a velocity of zero and reverses direction, after crossing through zero velocity the axis will apply the acceleration target rate rather than the deceleration target rate. Specified acceleration and deceleration values must always be positive.

Note that if the deceleration target value is set to zero Juno will use the specified acceleration target value for the deceleration value.

Figure 7-1 illustrates a complex profile in which the specified velocity and the direction of motion changes several times.

7.1 Settable Parameters

To control Juno's internal profile generator three parameters are specified; the maximum velocity, the acceleration, and the deceleration. All of these parameters can be updated 'on the fly' to allow complex profile shapes to be created.

These parameters use an encoding denoted "X.Y" with X indicating the number of bits representing the integer portion and Y indicating the number of bits used to represent the fractional component.

Name	Format	Representation, Range & units
Velocity	16.16	signed 32 bits (-32,768 to +32,767.9998 counts/cycle/2^{16})
Acceleration	8.24	unsigned 32 bits (-128 to +127.99999994 counts/cycle^2/2^{24})
Deceleration	8.24	unsigned 32 bits (-128 to +127.99999994 counts/cycle^2/2^{24})

The host commands **SetVelocity**, **SetAcceleration**, and **SetDeceleration** load the specified target profile values. The commands **GetVelocity**, **GetAcceleration**, and **GetDeceleration** retrieve them.

Specified target profile parameters are applied immediately. Whether or not these new parameters result in an immediate change depends on the profile being drawn. For example if a new deceleration value is programmed while the axis is accelerating this new deceleration value will not be applied until the profile enters a deceleration phase.

To query the instantaneous commanded profile values, use the commands **GetCommandedPosition**, **GetCommandedVelocity**, and **GetCommandedAcceleration**.

Example

Juno's profile generator will be used as the command source for the velocity loop. A profile that achieves a velocity of 135,000 encoder counts/sec is desired after an acceleration phase of 120 mSecs. Juno's cycle time has been left at the default value of 102.4 µSecs/cycle.

To convert counts/sec to counts/cycle we use: V (in counts/cycle) = V (in counts/sec) / (1.0 sec / .0001024 cycles/sec). Plugging in, we get V = 135,000 counts/sec / 9,765 cycles/sec = 13.825 counts/cycle. To scale this command to the 16.16 format used for the Juno target velocity we multiply by 2^{16} giving a 32-bit target velocity command of 13.825 * 65,536 = 906,035.

The acceleration command must achieve the desired velocity in .120 sec * 9,765 cycles/sec = 1,172 cycles. Therefore the acceleration is 13.825 counts/cycle / 1,172 cycle = .01180 counts/cycle^2. To scale this command to the 8.24 format used for the Juno target acceleration we multiply by 2^{24} giving a 32-bit acceleration command value to send of .01180 * 16,777,216 = 197,971.

7.2 Programmed Stops

Juno provides an event action mechanism that allows automatic control of the motor axis under various conditions such as motion error, over temperature condition, and disable. While some event action responses result in a complete disabling of the profile generator, others allow programmed stops of the motor thereby enabling safe deceleration of the motor to a stop.

Juno provides two types of controlled stops; a smooth stop and an abrupt stop. In a smooth stop the motor profile decelerates at the user-specified deceleration value until it reaches a velocity of zero. In an abrupt stop the velocity is instantaneously set to zero without a deceleration phase.

When the profile generator is active both a smooth stop and an abrupt stop will result in the user specified target velocity register being set to zero. This means that to restart a profile the target velocity has to be reloaded. For an abrupt stop, in addition to the target velocity being set to zero the instantaneous commanded velocity it also set to zero.

Usually event actions occur as the result of some autonomous system event such as a motion error. It is also possible however to specify that an event action occurs immediately using the **SetEventAction** command. This is a useful way for the host to execute smooth or abrupt stops directly via host commands.

For more information on setting up and recovering from event actions refer to Section 8.3, "Event Action Processing."

It is the responsibility of the user to insure that event actions responses, smooth or abrupt stops, and all specified profile parameters result in safe motion of the controller motor or actuator.

7.3 Enabling and Disabling the Profile Generator Module

Although it is not the same type of control module as the loop control modules the profile generator can be similarly enabled and disabled via the **SetOperatingMode** command. If disabled, the current commanded position will remain at its present value and the commanded velocity is set to zero.

In addition to manually disabling this module, there are event-related actions that may result in the profile generator module being automatically disabled. See Section 8.3, "Event Action Processing" for details.

A previously disabled profile generator module may be re-enabled in a number of ways. If the module was disabled using the **SetOperatingMode** command, then another **SetOperatingMode** command may be issued. If the module was disabled as part of an automatic event-related action then the command **RestoreOperatingMode** is used.

7.4 Profile Generator as Loop Command Source

Despite the fact that the profile control parameters are referred to as accelerations and velocities, depending on how the profile generator output is used the actual profiled command value may be an outer loop quantity such as pressure, velocity, current, or voltage. The following table shows which profile generator registers are utilized for each Juno control loop when profile generator is selected as the command source:

Control Loop	Profile Generator Register	Commanded Value	Format
Position loop	Position	Position	32 bits
Outer loop	Velocity	Outer loop quantity	32 bits
Velocity loop	Velocity	Velocity	32 bits
Current loop	Velocity	Current	32 bits
Motor output	Velocity	Voltage	32 bits

For more information on selecting the profile generator as the Juno control loop source and for detailed information on command scaling refer to Chapter 3, *Position/Outer Loop*, Chapter 4, *Velocity Loop*, Chapter 5, *Current Loop*, and Chapter 6, *Motor Output*.

8. Motion Monitoring & Control

8

In This Chapter

- Position Tracking
- Status Registers
- Event Action Processing
- FaultOut Signal
- Trace
- Host Interrupts

8.1 Position Tracking

Juno ICs utilize position feedback to calculate the motor velocity, for safety checking, or for commutation. The most common feedback method is quadrature encoding however Juno can also use Hall sensors, if provided, for position feedback. Whether quadrature or Hall signals are used, Juno continually monitors the position feedback signals and accumulates a 32-bit position value called the actual position. At power-up, the default actual position is zero.

The actual position can be set with one of two commands: **SetActualPosition**, and **AdjustActualPosition**. The current actual position can be retrieved using the command **GetActualPosition**. **SetActualPosition** sets the position to an absolute specified 32-bit value. **AdjustActualPosition** changes the current actual position by a signed relative value. For example a value of –25 specified using this command will subtract 25 from the current actual position.

In addition to retrieving the actual axis position, it is also possible to retrieve an estimation for the instantaneous velocity from the position feedback of the axis. This is accomplished using the command **GetActualVelocity**. Note that the provided velocity is an estimated quantity, created by subtracting the current position from the previous cycle's position. It is therefore subject to jitter or noise, particularly at low velocity.

8.1.1 Position Wraparound

The 32-bit actual position register continually tracks the axis position using either Juno's quadrature encoder signal inputs or Hall sensor signal inputs. The full range of trackable positions is -2,147,483,647 to +2,147,483,647.

In the event that a spinning axis exceeds either of these position limits the actual position wraps around, with the largest positive position wrapping around to become the smallest negative position, and the smallest negative position wrapping around to become the largest positive number. When such a wraparound occurs a corresponding bit in the Event Status register is set.

Position wraparound, should it occur during operation, is generally not a consequential event. In the case that Juno is used for torque control or velocity control a position wraparound will have no impact on the behavior of the axis, nor is there a limit to the number of such wraparound events that may occur. Users may want to be aware of a wraparound if they are directly reading the 32-bit actual position for purposes of internal comparison or safety checking.

In addition to the actual position register all of Juno's position based registers such as the commanded profile position similarly automatically wrap, however only a wraparound of the actual position triggers the corresponding event status flag being set.

For more information on the Event Status register see Section 8.2.1, "Event Status Register."

8.1.2 Quadrature Encoder Input

Incremental encoder feedback utilizes two signal inputs: *QuadA* and *QuadB*. There is also an optional *Index* signal input, which normally indicates when the motor has made one full rotation. The A and B signals encode square wave inputs offset from each other by 90°.

To specify quadrature encoders for the position feedback source the command **SetEncoderSource** is used. A value of 'none' can also be programmed indicating that there is no position feedback connected. To read the selected encoder source back the command **GetEncoderSource** is used.

For complete information on interfacing to quadrature encoder signals refer to the *MC78113 Electrical Specifications*.

8.1.3 Quadrature Position Capture

Juno ICs support a high-speed position capture register that allows the current axis location (as determined by the attached encoder) to be captured when triggered by the *Index* signal. When a capture is triggered, the contents of the actual position registers are transferred to the position capture register, and the capture-received indicator (bit 3 of the Event Status register) is set.

To read the capture register, the command **GetCaptureValue** is used. The capture register must be read before another capture can take place. Reading the position capture register causes the trigger to be re-armed, allowing for more captures to occur. As for all Event Status register bits, the position capture indicator may be cleared by using the command **ResetEventStatus**.

8.1.4 Hall Sensor Input

Juno provides three pins for direct Hall sensor input. Hall sensors are generally only used with Brushless DC motors and are used for commutation or commutation initialization.

In addition to commutation however, Hall inputs can also be used as the source for position feedback input. In this mode Juno continually tracks the forward or backward progression of the motor axis using the Hall state transitions. For more information on the function of Hall sensors and the Hall signal sequence that Juno expects for commutation and position tracking refer to Section 9.1, "Hall-Based Commutation."

To specify Hall sensors for the position feedback source the command **SetEncoderSource** is used. To read this value back the command **GetEncoderSource** is used.

The capture facility discussed in Section 8.1.3, "Quadrature Position Capture" does not function when the encoder source is set to Halls. It only functions with the encoder source set to quadrature.

8.2 Status Registers

There are five bit-oriented status registers that provide a continuous report on the state of Juno and the controlled axis. These five 16-bit registers are Event Status, Activity Status, Drive Status, Drive Fault Status, and Signal Status.

8.2.1 Event Status Register

The Event Status register is designed to record events that do not continuously change in value but rather tend to occur once due to a specific event. As such, each bit in this register is set by the MC78113 and cleared by the host.

The Event Status register is defined in the following table:

Bit	Name	Description
0	*Reserved*	*May contain 0 or 1.*
1	Position wraparound	Set when the actual motor position exceeds 7FFF FFFFh (the most positive position), and wraps to 8000 0000h (the most negative position), or vice versa.
2	*Reserved*	*May contain 0 or 1*
3	Capture received	Set when the high-speed position capture hardware acquires a new position value.
4	Motion error	Set when the actual position differs from the commanded position by an amount more than the specified maximum position error.
5-6	*Reserved*	*May contain 0 or 1*
7	Instruction error	Set when an instruction error occurs.
8	Disable	Set when the user disables MC 78113 by making the enable signal inactive.
9	Overtemperature fault	Set when an overtemperature fault occurs.
10	Drive Exception	Set when one of a number of drive exceptions, such as bus overvoltage or undervoltage fault occurs.
11	Commutation error	Set when a commutation error occurs.
12	Current foldback	Set when current foldback occurs.
13	Run time error	Set when a runtime error occurs.
14-15	*Reserved*	*May contain 0 or 1.*

The command **GetEventStatus** returns the contents of the Event Status register.

Bits in the Event Status register are latched. Once set, they remain set until cleared by a host instruction or a system reset. Event Status register bits may be reset to 0 by the instruction **ResetEventStatus**, using a 16-bit mask. Register bits corresponding to 0s in the mask are reset; all other bits are unaffected.

The Event Status register may also be used to generate a host interrupt signal using the **SetInterruptMask** command. See Section 8.2.1, "Event Status Register" for more information.

8.2.1.1 Instruction Error

Bit 7 of the Event Status register indicates an instruction error. Such an error occurs when an incorrect opcode is sent, when an argument is out of bounds, or when some other instruction error occurs.

Should an instruction error occur, the invalid parameters are ignored, and the **Instruction Error** indicator of the Event Status register is set.

To determine the nature of the instruction error a **GetInstructionError** command is sent. This command indicates the nature of the instruction error including when instruction errors occur during processing of initialization commands in NVRAM. The returned word provides the first two errors recorded after the previous execution of the **GetInstructionError** command.

8.2.2 Activity Status Register

Activity Status register bits are not latched, they are continuously set and reset to indicate the status of the corresponding conditions.

The Activity Status register is defined in the following table:

Bit	Name	Description
0	Phasing initialized	Set 1 when the motor's commutation hardware has been initialized. Cleared 0 if not yet initialized.
1	At maximum velocity	Set 1 when the commanded velocity is equal to the maximum velocity specified by the host. Cleared 0 if it is not. This bit only functions in conjunction with the profile generator and is not set if the loop command source is set to anything other than profile generator.
2-8	*Reserved*	*May contain 0 or 1.*
9	Position capture	Set 1 when a new position value is available to read from the high speed capture hardware. Cleared 0 when a new value has not yet been captured. While this bit is set, no new values will be captured. The command **GetCaptureValue** retrieves a captured position value and clears this bit, thus allowing additional captures to occur.
10	In-motion indicator	Set 1 when the trajectory profile commanded position is changing. Cleared 0 when the commanded position is not changing.
11-15	*Reserved*	*May contain 0 or 1.*

The command **GetActivityStatus** returns the contents of the Activity Status register for the specified axis.

8.2.3 Drive Status Register

The Drive Status register functions similarly to the Activity Status register in that it is not latched. Status bits are continuously set and reset to indicate the status of the corresponding conditions. The specific status bits provided by the Drive Status register are defined in the following table:

Bit	Name	Description
0	Calibration completed	Set 1 when an analog input calibration procedure is completed. Cleared if not completed.
1	In foldback	Set 1 when in foldback, cleared 0 if not in foldback.
2	Overtemperature	Set 1 when the axis is currently in an overtemperature condition. Cleared 0 if the axis is currently not in an overtemperature condition.
3	Shunt Active	Set 1 when shunt request is active. Cleared 0 if not.
4	In holding	Set 1 when the axis is in a holding current condition, cleared 0 if not.
5	Overvoltage	Set 1 when the axis is currently in an overvoltage condition. Cleared 0 if the axis is currently not in an overvoltage condition.
6	Undervoltage	Set 1 when the axis is currently in an undervoltage condition. Cleared 0 if the axis is currently not in an undervoltage condition.
7-11	*Reserved*	*May contain 0 or 1.*
12	Output Clipped	Set 1 when the amplifier current command can not be met because of output clipping. This information is used by Juno to prevent PID saturation.
13	*Reserved*	*May contain 0 or 1.*
14	Initializing	Set 1 when MC 78113 is initializing from NVRAM commands. Set 0 when initializing is complete.
15	*Reserved*	*May contain 0 or 1.*

The command **GetDriveStatus** returns the contents of the Drive Status register for the specified axis.

8.2.4 Drive Fault Status Register

To simplify recovery from an electrical fault, as well as to read other drive-related faults, Juno provides a Drive Fault Status register that can be read using the command **GetDriveFaultStatus**. The bits in this register operate similarly to the bits in the Event Status register in that they are set by Juno, and cleared by the host. The command that is used to clear the bits in this register is **ClearDriveFaultStatus**. Note that this command only clears bits that were indicated as set by a previous **GetDriveFaultStatus** command. The following table indicates the contents of the Drive Fault Status register:

Bit	Name	Description
0	Overcurrent	Set 1 to indicate a fault due to a short circuit or overload in the drive output.
1-4	*Reserved*	*May contain 0 or 1*
5	Overvoltage	Set 1 to indicate an overvoltage condition of the external bus voltage input.
6	Undervoltage	Set 1 to indicate an undervoltage condition of the external bus voltage input.
7	*Reserved*	*May contain 0 or 1*
8	Foldback	Set 1 to indicate that a current foldback event has occurred.
9, 10	*Reserved*	*May contain 0 or 1*
11	Watchdog	Set 1 to indicate that a watchdog event has occurred.
12	*Reserved*	*May contain 0 or 1*
13	Brake	Set 1 to indicate that the Brake signal input pin has gone active.
14, 15	*Reserved*	*May contain 0 or 1*

8.2.5 Signal Status Register

The Signal Status register provides real-time signal levels for various Juno IC I/O pins. The Signal Status register is defined in the following table:

Bit	Name	Description
0	A encoder	A signal of quadrature encoder input.
1	B encoder	B signal of quadrature encoder input.
2	Index encoder	Index signal of quadrature encoder input.
3-6	*Reserved*	*May be 0 or 1.*
7	HallA/ AtRest	Hall effect sensor input number A or *AtRest* signal.
8	HallB	Hall effect sensor input number B.
9	HallC	Hall effect sensor input number C.
10	*Reserved*	*May contain 0 or 1.*
11	Pulse	Pulse signal input.
12	*Reserved*	*May contain 0 or 1.*
13	/Enable	Enable signal input.
14	FaultOut	Fault signal output.
15	Direction	Direction signal input

The command **GetSignalStatus** returns the contents of the Signal Status register for the specified axis. All Signal Status register bits are inputs except bit 14 (*FaultOut*).

The input bits in the Signal Status register represent the actual hardware signal level combined with the state of the signal sense mask described in the next section. That is, if the signal level is high, and the corresponding signal mask

bit is 0 (do not invert), then the bit read using **GetSignalStatus** will be 1. Conversely, if the signal mask for that bit is a 1 (invert), then a high signal on the pin will result in a read of 0 using the **GetSignalStatus** command.

The output bits in the Signal Status register are not affected by the signal sense mask. For these signals a 1 indicates an active condition and a 0 indicates a non active condition.

8.2.5.1 Signal Sense Mask

The bits in the Signal Status register represent the high/low state of various signal pins. It is possible to invert the incoming signal by host command to match the signal interpretation of the user's hardware. This function is accessed via the command **SetSignalSense**, and can be read back using the command **GetSignalSense**.

The default value of the signal sense mask is "not inverted" except for the *Index* signal, which has a default value of "inverted." The bits of the signal sense mask register are defined in the following table:

Bit	Name	Interpretation
0	A encoder	Set 1 to invert quadrature A input signal. Clear 0 for no inversion.
1	B encoder	Set 1 to invert quadrature B input signal. Clear 0 for no inversion.
2	Index encoder	Set 1 to invert, clear 0 for no inversion. This means that for active low interpretation of index signal, set to 0; and for active high interpretation, set to 1.
3-6	*Reserved*	
7	Hall A/AtRest	Set 1 to invert HallA or AtRest signal. Clear 0 for no inversion.
8	Hall B	Set 1 to invert HallB signal. Clear 0 for no inversion.
9	Hall C	Set 1 to invert HallC signal. Clear 0 for no inversion.
10	AxisOut	Set 1 to invert AxisOut signal. Clear 0 for no inversion.
11	Pulse	Set 1 to define active transition as low-to-high. Clear 0 to define active transition as high-to-low.
12	Motor Direction	Set 1 to invert Motor Direction. Clear 0 for no inversion.
13–14	*Reserved*	
15	Direction	Set to 1 to invert Direction signal. Clear 0 for no inversion.

8.3 Event Action Processing

Juno ICs provide a programmable mechanism for automatically reacting to various safety or performance-related events. This mechanism is called event action processing.

Each monitored event condition may have an associated event action defined for it. The following table lists each of the event conditions that Juno monitors along with the default event actions that will occur if no user specified event actions are provided. Unless otherwise noted all event actions are allowed for each event:

Condition Name	Default Action	Description
Immediate	Not Applicable	Occurs immediately upon being specified. This event allows the host to explicitly execute an event action
Motion error	Disable Motor Output	Occurs when a motion error condition is detected
Current foldback	Disable Motor Output	Occurs when the amplifier current output goes into a foldback condition.
Encoder Position Capture	No Action	Occurs when the quadrature encoder Index signal has triggered a position capture
Overtemperature	Disable Motor Output	Occurs when an overtemperature condition is detected
Disabled	Disable Motor Output	Occurs when the Enable signal goes inactive. The programmed event action must be Disable Motor Output or Brake

Condition Name	Default Action	Description
Commutation error	No Action	Occurs when a commutation error is detected. The programmed event action must be no action, Disable Motor Output, or Brake
Overcurrent	Disable Motor Output	Occurs when an overcurrent condition is detected. The programmed event action must be Disable Motor Output or Brake
Overvoltage	Disable Motor Output	Occurs when an overvoltage condition is detected
Undervoltage	Disable Motor Output	Occurs when an undervoltage condition is detected
Watchdog timeout	No Action	Occurs when a watchdog timeout condition is detected
Brake signal	Brake	Occurs when the Brake signal goes active. The programmed event action must be Disable Motor Output or Brake

The following table describes the event actions that can be programmed:

Action Name	Description
No Action	No action taken.
Smooth Stop	Causes a smooth stop to occur at the current active deceleration rate.
Abrupt Stop	Commands an instantaneous halt of the motor.
Disable Position/Outer Loop	Disables profile generator and position/outer loop module.
Disable Velocity Loop	Disables profile generator, position/outer loop, and velocity loop.
Disable Current Loops	Disables profile generator, position/outer loop, velocity loop, and current loop modules.
Disable Motor Output	Disables profile generator, position/outer loop, current loop, velocity loop, and motor output modules.
Brake	Turns the brake function on and disables the profile generator, position/outer loop, velocity loop, current loop, and motor output module.

8.3.1 Event Processing

Upon powerup and initialization completion Juno begins to continuously monitor the event conditions and executes the programmed event action if they occur. When the programmed action is executed, related actions may occur such as setting the appropriate bit in the Event Status register.

To recover from an event action, the cause of the event occurring should be investigated and corrected. In cases where an event status bit was set this flag should be reset using the **ResetEventStatus** command. The command **RestoreOperatingMode** is then used to restore the operating mode previously specified using **SetOperatingMode** command. Note that if the event condition is still present, then the event action will immediately occur again.

It is the responsibility of the user to safely and thoroughly investigate the cause of event-related events, and only restart motion operations when appropriate corrective measures have been taken.

If the event action programmed was either *No Action, Abrupt Stop*, or *Smooth Stop*, then the **RestoreOperatingMode** command will have no effect. It is intended to restore disabled modules only, and has no effect on profile generator parameters.

Once programmed, an event action will be in place until reprogrammed. The occurrence of the event condition does not reset the programmed event action.

8.3.2 Automatic Event Recovery

In order to facilitate easier recovery from safety-related faults such as overtemperature or current foldback an automatic recovery process is available. This mode is most often used when operating Juno without a host and in conjunction with the *FaultOut* signal, but it may be selected even when host communications are available. Automatic event recovery mode is activated using the **SetDriveFaultParameter** command, and may be read back using **GetDriveFaultParameter**.

To program the *FaultOut* signal to go active upon occurrence of various programmable conditions the **SetFaultOutMode** command is used. After the *FaultOut* signal goes active, the external logic must delay a minimum of 150 µSec, but thereafter may request that Juno attempt to automatically recover by deasserting, and then asserting, the *Enable* signal. The *Enable* signal must be in the deasserted state for at least 150 µSec for the request to be recognized.

When an automatic recovery request is recognized by Juno it behaves as though the command sequence **ResetEventStatus 0** and **RestoreOperatingMode** has been sent to it by a host controller. As is the case when these commands are sent by the host controller, if the fault condition is still present when recovery is attempted, Juno will immediately again disable itself, and a recovery procedure must once again be requested. If the fault has been corrected however a recovery request will result in resumption of normal Juno operation.

8.4 FaultOut Signal

Juno's *FaultOut* signal is used to indicate an occurrence of one or more of the faults indicated in the previous sections as well as other fault conditions. This signal is always active high. Its sense cannot be changed using the command **SetSignalSense**. Any bit condition of the Event Status register may be used to trigger activation of the fault signal. This is done using the command **SetFaultOutMask**. The value set using this command can be read back using **GetFaultOutMask**.

The bit conditions specified via this mask are logically ANDed with the Event Status register. Any resultant non-zero value will cause the *FaultOut* signal to go active. See Section 8.2.1, "Event Status Register" for information on the Event Status register.

Example

Programming **SetFaultOutMask** with a value of 1,040 (0x410) configures the *FaultOut* signal to be driven high upon a motion error (bit #4) or a drive exception error (bit #10).

8.5 Trace

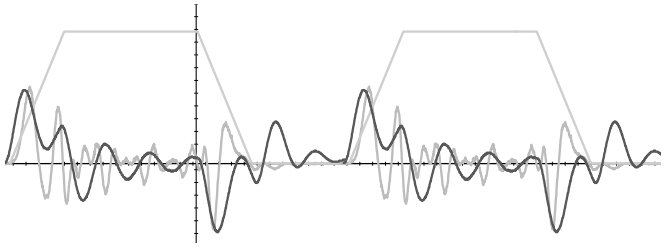


Figure 8-2: Example Motion Trace Capture

Trace is a powerful feature of the Juno ICs that allows various parameters and registers to be continuously captured and stored to an on-chip RAM buffer. The captured data may later be downloaded by the host. Traces are useful for optimizing servo performance, verifying trajectory behavior, capturing sensor data, or to assist with any type of monitoring where a precise time-based record of the system's behavior is required.

Trace data capture (by the Juno IC) and trace data retrieval (by the host) are executed as two separate processes. The host specifies which parameters will be captured, and how the trace will be executed. Then Juno performs the trace. Finally, the host retrieves the data after the trace is complete. It is also possible to perform continuous data retrieval, even as Juno is continuing to collect additional trace data.

To start a trace, the host must specify a number of parameters. They are:

Parameter	Description
Trace period	Juno can capture the value of the trace variables for every single cycle, every other cycle, or at any specified period.
Trace variables	There are many dozens of separate variables and registers within the Juno IC that may be traced; for example, actual position, Event Status register, position error, etc. The user must specify the variables from which data will be recorded.
Trace mode	Juno can trace in one of two modes: one-time, or rolling mode. This determines how the data is stored, and whether the trace will stop automatically or whether it must be stopped by the host.
Trace start/stop conditions	To allow precise synchronization of data collection, it is possible to define the start and stop conditions for a given trace. Juno monitors these specified conditions and starts or stops the trace automatically without host intervention.

8.5.1 Trace Period

The tracing system supports a configurable period register that defines the frequency at which data is stored to the trace buffer. The tracing frequency is specified in units of cycle times.

The command **SetTracePeriod** sets the trace period, and the command **GetTracePeriod** retrieves it.

8.5.2 Trace Variables

When traces are running, one to four data variables may be stored to the trace buffer at the same time. The four trace variable registers are used to define which parameters are stored. The following commands are used to configure the trace variables.

The command **SetTraceVariable** selects which traceable parameter will be stored by the trace variable specified. The command **GetTraceVariable** retrieves this same value.

More than 50 variables are available for trace on Juno ICs. For a complete list of traceable variables please refer to the *Juno Velocity & Torque Control IC Programming Reference*.

8.5.3 Trace Modes

As trace data is collected, it is written to sequential locations in the trace buffer. When the end of the buffer is reached, the trace mechanism will behave in one of two ways, depending on the selected mode.

If one-time mode is selected, then the trace mechanism will stop collecting data when the buffer is full.

If rolling-buffer is selected, then the trace mechanism will wrap around to the beginning of the trace buffer and continue storing data. Data from previous cycles will be overwritten by data from subsequent cycles. In this mode, the diagnostic trace will not end until the conditions specified in a **SetTraceStop** command are met.

Use the command **SetTraceMode** to select the trace mode. The command **GetTraceMode** retrieves the trace mode.

8.5.4 Trace Start/Stop Conditions

The command **SetTraceStart** is used to specify the conditions that will cause the trace mechanism to start collecting data. A similar command, **SetTraceStop**, is used to define the condition that will cause the trace mechanism to stop collecting data. Both **SetTraceStart** and **SetTraceStop** allow one of several trigger types to be programmable as indicated in the following table:

ID	Name	Description
0	Immediate	This trigger type indicates that the trace starts (stops) immediately when the SetTraceStart (SetTraceStop) command is issued. If this trigger type is specified, the trigger axis, bit number, and bit state value are not used.
2	Event Status	The specified bit in the Event Status register will be constantly monitored. When that bit enters the defined state (0 or 1), then the trace will start (stop).
3	Activity Status	The specified bit in the Activity Status register will be constantly monitored. When that bit enters the defined state (0 or 1), then the trace will start (stop).
4	Signal Status	The specified bit in the Signal Status register will be constantly monitored. When that bit enters the defined state (0 or 1), then the trace will start (stop).
5	Drive Status	The specified bit in the Drive Status register will be constantly monitored. When that bit enters the defined state (0 or 1), then the trace will start (stop).

8.5.5 Downloading Trace Data

Once a trace has executed and the trace buffer is full (or partially full) of data, the captured data may be downloaded by the host using the standard commands to read from the external buffer memory. See Section 8.5.7, "Buffer Commands" for a complete description of memory buffer commands.

The command **GetTraceCount** is used to get the number of 32-bit words of data stored in the trace buffer. This value may be used to determine the number of **ReadBuffer** commands that must be issued to download the entire contents of the trace buffer.

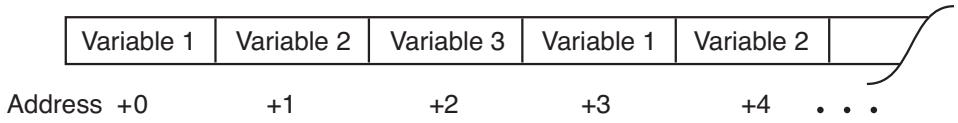


Figure 8-3: Trace Data Format

During each trace period, each of the trace variables is used in turn to store a 32-bit value to the trace buffer. Therefore, when data is read from the buffer, the first value read would be the value corresponding to trace variable 1, the second value will correspond to trace variable 2, up to the number of trace variables used. Figure 8-3 show this, illustrating an example trace buffer when 3 variables were specified for trace.

8.5.6 Trace & NVRAM Memory Buffers

Juno ICs are capable of accessing memory for the retrieval of trace data stored in RAM or in connection with the NVRAM.

These memory operations utilize buffers, which are contiguous blocks of memory with a defined start location, size, and access ID #. Juno provides two pre-defined buffers, a RAM trace buffer which has a start address of 0x0000 0000, a size of 6,144 16-bit words, and an ID # of 0. Juno's NVRAM buffer has a start address of 0x2000 0000, a size of 1,024 16-bit words, and an ID# of 1.

Central to accessing buffers are the read index and the write index. The read index may be assigned a value between 0 and L-1 (where L is the buffer length). It defines the location from which the next value will be read. When a value is read from the memory buffer, the read index is automatically incremented, thus selecting the next value for reading.

Similarly, the write index ranges from 0 to L-1 and defines the location at which the next value will be written. The write index is incremented whenever a value is written to a buffer. The default value for both the read and write indexes are 0.

If either the read or the write index reaches the end of the buffer, it is automatically reset to 0 on the next read/write operation.

8.5.7 Buffer Commands

The table below details host commands that access, and monitor buffers.

Command	Argument	Description
GetBufferStart	bufferID	Returns the base address of the specified buffer.
GetBufferLength	bufferID	Returns the length of the specified buffer.
SetBufferReadIndex	bufferID, index	Sets the read index for the specified buffer. index is a 32-bit integer in the range 0 to length–1, where length is the current buffer length.
GetBufferReadIndex	bufferID	Returns the value of the read index for the specified buffer.
SetBufferWriteIndex	bufferID, index	Sets the write index for the specified buffer. index is a 32-bit integer in the range 0 to length–2, where length is the current buffer length.
GetBufferWriteIndex	bufferID	Returns the value of the write index for the specified buffer.
ReadBuffer	bufferID	Returns a 32-bit value from the specified buffer. The location from which the value is read is determined by adding the base address to the read index. After the value has been read, the read index is incremented. If the result is equal to the current buffer length, the read index is set to zero (0).
ReadBuffer16	bufferID	Returns a 16-bit value from the specified buffer. This command is otherwise identical to the ReadBuffer command.
WriteBuffer	bufferID, value	Writes a 32-bit value to the specified buffer. The location to which the value is written is determined by adding the base address to the write index. After the value has been written, the write index is incremented. If the result is equal to the current buffer length, the write index is set to zero (0).

The trace buffer (BufferID 0) may be written to or read from by the user as desired. The NVRAM buffer (BufferID1) may not be written to by buffer commands but may be read from via buffer commands. For information on writing to the Juno NVRAM buffer refer to Section 12.4, "Non-Volatile (NVRAM) Storage."

8.6 Host Interrupts

Interrupts allow the host to become aware of a special Juno condition without the need for continuous monitoring or polling of the status registers. For this purpose Juno provides a physical ***HostInterrupt*** signal that is generally connected to an interrupt input on the microprocessor. For more information on this ***HostInterrupt*** signal, see the *MC 78113 Electrical Specifications.*

If CANbus communications are being used host interrupt events also cause a special CAN message to be sent. See Section 13.4, "Controller Area Network (CAN)" for more information on CANbus communications.

The events that trigger a host interrupt are the same as those that set the assigned bits of the Event Status register. Using a 16-bit mask, the host may condition any or all of these bits to cause an interrupt. This mask is set using the command **SetInterruptMask**. The value of the mask may be retrieved using the command **GetInterruptMask**. The mask bit positions correspond to the bit positions of the Event Status register. If a 1 is stored in the mask, then a 1 in the corresponding bit of the Event Status register will cause an interrupt to occur. See Section 8.2.1, "Event Status Register" for details.

Juno continually and simultaneously scans the Event Status register and interrupt mask to determine if an interrupt has occurred. When an interrupt occurs, the *HostInterrupt* signal is made active. At this point, the host can respond to the interrupt but is not required to do so.

To process the interrupt, normal Juno commands are used. The specific commands sent by the host to process the interrupt depend on the nature of the interrupting condition. At minimum, the interrupting bit in the Event Status register should be cleared using the **ResetEventStatus** command. If this is not done then the same interrupt will immediately occur once interrupts are re-enabled.

Once the host has completed processing the interrupt it should send a **ClearInterrupt** command to clear the interrupt line and re-enable interrupt processing. Note that if another interrupt is active the interrupt line will only be cleared momentarily and then reasserted.

Example

The following provides a typical sequence of interrupts and host responses. In this example, a motion error has occurred causing an abrupt stop. In this example, the interrupt mask has been set so that motion errors will cause an interrupt.

Event	Host action
Motion error generates interrupt.	
	Sends **GetEventStatus** instruction, detects that motion error flag is set.
	Issues a **ResetEventStatus** command to clear.
	Returns the axis to closed loop mode by issuing a **RestoreOperatingMode** command.
	Issues a **ClearInterrupt** command to reset the interrupt signal.
Juno IC clears motion error bit and disables host interrupt line.	

At the end of this sequence, all status bits are cleared and the interrupt line is inactive.

9. Brushless DC Motor Control

In This Chapter

Juno ICs provide a number of special features for supporting Brushless DC motors. These include input of Hall sensors, multi-phase signal generation, and a number of other capabilities related to phasing control and commutation.

To drive a brushless DC motor the motor's rotor angle must be known as it continually changes. This is accomplished using one of two methods. The first is by using Hall sensors, and the second is by using the quadrature position encoder. In both cases these sensors must be directly connected to the motor shaft. Generally speaking, if both Hall signals and encoder signals are available the encoder should be used for commutation as it will provide smoother motion and higher overall performance than Hall sensors.

To select whether the phasing of the motor will be Hall-based or encoder-based, the command **SetCommutationMode** is used. The value set can be read back using the command **GetCommutationMode**.

9.1 Hall-Based Commutation

The Hall sensor signals are input directly to Juno through the signals **HallA, HallB, and HallC.** To read the current status of the Hall sensors, use the command **GetSignalStatus**.

To accommodate varying types of Hall sensors, or sensors containing inverter circuitry, the signal level/logic interpretation of the Hall sensor input signals may be set through the host. The command **SetSignalSense** accepts a bit-programmed word that controls whether the incoming Hall signals are interpreted as active high or active low. To read this Hall interpretation value the command **GetSignalSense** is used.

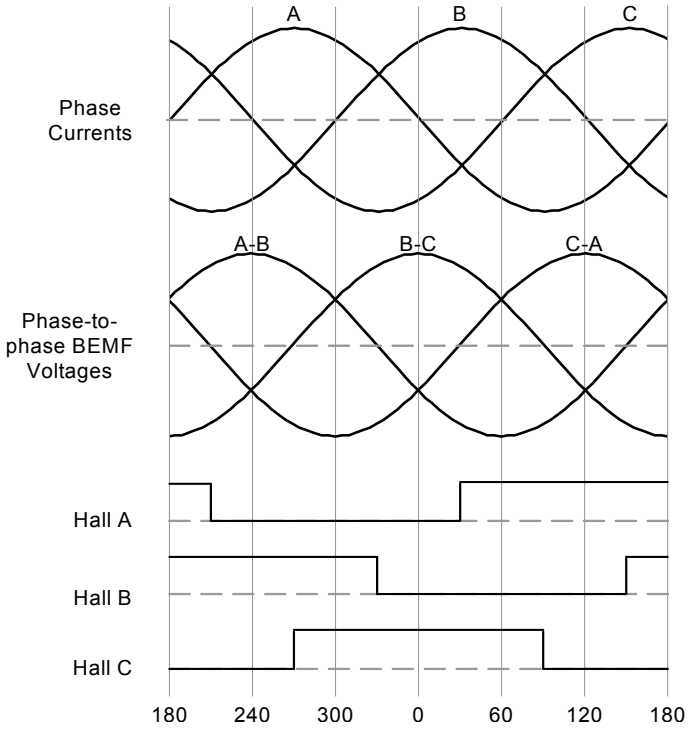
9.2 Encoder-Based Commutation

If the motor commutation will occur via an encoder the number of encoder counts per electrical cycle must be specified. This parameter indicates to Juno the number of encoder counts required to complete a single full electrical cycle. The number of electrical cycles can usually be determined from the motor manufacturer's specification and is exactly half the number of poles.

The command used to set the number of encoder counts per electrical cycle is **SetCommutationParameter**. To read this value back use the command **GetCommutationParameter.** The number of encoder counts per electrical cycle is not required to be an exact integer.

9.2.1 Phase Initialization

Figure 9-1: Hall-based Phase Initialization



When commutating via the encoder, in addition to specifying the counts per electrical cycle Juno must determine the proper initial phase angle of the motor relative to the encoder position. This information is determined using a procedure called phase initialization. Note that a phase initialization procedure is not necessary if Hall-based commutation is selected.

Juno provides three methods to perform phase initialization: Hall-based, pulse phase, and direct-set. Each are described in detail in the subsequent sections.

9.2.1.1 Hall-Based Phase Initialization

The most common and the simplest method of encoder phase initialization is Hall-based. In this mode, three Hall sensor signals are used to determine the motor phasing. Encoder-based commutation begins automatically after the motor has moved through at least one Hall state transition.

Figure 9-1 illustrates the relationship between the state of the three Hall sensor inputs, the sinusoidally commutated phase current commands, and the motor phase-to-phase back EMF waveforms during forward motion of the motor. A Hall to back EMF phasing diagram is a common way to specify the required alignment and a such a diagram is often provided by the motor supplier.

Juno expects 120-degree separation between Hall signal transitions. To commutate using Hall sensors located 60 degrees apart, swap and invert the appropriate Hall signals and motor phases to generate the expected Hall states. Note that the command **SetSignalSense** can be used to accomplish hall signal inversion.

To set Juno for Hall-based initialization, use the command **SetPhaseInitializeMode**. Initialization is performed using the command **InitializePhase**.

9.2.1.2 Pulse Phase Initialization

Juno provides the ability to initialize encoder-based phasing of Brushless DC motors without the need for explicit phasing signals such as Hall sensors using a proprietary technique called pulse phase initialization. Pulse phase initialization executes by applying a rapid burst of positive/negative command pulse pairs followed by a longer ramp pulse. As long as the motor motion is unencumbered through this sequence the resulting phase initialization should be accurate and repeatable.

To set Juno for pulse phase initialization use the command **SetPhaseInitializeMode**.

The following table shows the parameters that are set in association with pulse phase initialization:

Parameter	Range & Units	Description
PositivePulseTime	0-32,767 cycles	Is the time that the positive portion of the burst pulse is applied
NegativePulseTime	0-32,767 cycles	Is the time that the negative portion of the burst pulse is applied
PulseMotorCommand	1-32,767 % / 327.67	Is the motor command that will be applied during the burst mode of the initialization procedure. The specified value represents a % motor command from 0 to 100%.
RampMotorCommand	1-32,767 % / 327.67	Is the motor command that is applied during the ramp portion of the initialization procedure. The specified value represents a % motor command from 0 to 100%.
RampTime	0-32,767 cycles	Is the time that the slow ramp and hold is applied.

To set any of these five 16-bit parameters the command **SetPhaseParameter** is used. **GetPhaseParameter** is used to read these same values back.

After these parameters are loaded in an **InitializePhase** command is sent. While varying somewhat with motor size, typical total durations of the pulse phase sequence is 500 mSec. A flag in the Activity Status register indicates whether phase initialization has completed.

Setting up appropriate pulse phase parameters for various applications can most easily be accomplished with PMD's Pro-Motion software. To perform your own determination of pulse phase parameters refer to the *Juno Velocity & Torque Control IC Programming Reference.*

Pulse phase initialization can only function properly if motor movement is free and unencumbered.

9.2.1.3 Direct-Set Phase Initialization

If, after power-up, the location of the motor phasing is known, the phase angle can be directly set using the **SetCommutationParameter** command. This typically occurs when sensors such as resolvers are used where the returned motor position information is absolute in nature. Note that when using this approach the **InitializePhase** command is not used.

9.2.2 Automatic Phase Correction

To enhance commutation reliability the Juno ICs provide the ability to automatically correct the commutation phase during encoder-based commutation. Note that if Hall-based commutation is used, this feature is not necessary. Either an index signal or Hall signals can be used for this automatic correction function.

9.2.2.1 Index-Based Phase Correction

To utilize automatic phase correction the motor encoder provides an index pulse signal to Juno once per rotation. Index phase referencing is recommended for all rotary brushless motors with quadrature encoders directly mounted on the motor shaft. For linear brushless motors, it is generally not used. However, it may be used as long as the index pulses are arranged so that each pulse occurs at the same phase angle within the commutation cycle.

With an index signal properly installed, Juno will automatically adjust the commutation angle to correct for any small losses of encoder counts that may occur. If the loss of encoder counts becomes excessive, or if the index pulse does not arrive at the expected location within the commutation cycle, a commutation error occurs, which is indicated via bit 11 in the Event Status register. This bit is set if the required correction is greater than (PhaseCounts / 128) + 4.

To recover from a commutation error this bit should be cleared by the host. Depending on the cause of the error there may be no further errors or errors may continue to occur.

A commutation error may indicate a serious problem with the motion system, potentially resulting in unsafe motion. It is the responsibility of the host to determine and correct the cause of commutation errors.

9.2.2.2 Hall-Based Phase Correction

Hall signals may also be used for automatic commutation phase adjust when an index pulse is not available.

Hall-based phase error detection functions similarly to index-based phase correction. Bit 11 of the Event Status register signals a commutation error and recovery occurs in the same manner. However the absolute amount of error allowed before a commutation error occurs, normally Phase Counts / 128 + 4 with index-based phase correction, is Phase Counts/32 + 4.

To set the commutation phase correction mode to either Index or Hall-based use the command **SetPhaseCorrectionMode**. The command **GetPhaseCorrectionMode** reads back these same values.

9.2.3 Adjusting the Phase Angle

Juno supports the ability to change the motor's phase angle directly, both when the motor is stationary and when it is in motion. Although this is not generally required, it can be useful during testing, or during phase initialization when direct-set methods are used. Note that the phase angle can not be changed if Hall-based commutation is used.

To change the phase angle when the motor is stationary, use the command **SetCommutationParameter**. To change the phase angle while the motor is moving, the index pulse is required and a quantity known as the phase offset rather than the phase angle is adjusted. The phase offset, once set, takes effect only when an index pulse occurs. After phase initialization has occurred, the phase angle of the index pulse is stored in the phase offset register. This 32-bit offset register can be read using the command **GetCommutationParameter**.

For a given motor, the index pulse may be located anywhere within the commutation cycle, since it will usually vary in position from motor to motor. Only motors that have been mechanically assembled so that the index position is referenced to the motor windings will have a consistent index position relative to the commutation zero location.

Before phase initialization has occurred, the phase offset register will have a value of FFFF FFFFh. Once phase initialization has occurred and the motor has been rotated so that at least one index pulse has been received, the phase offset value will be stored as a positive number with a value between 0 and the number of encoder counts per electrical cycle (phase counts).

The phase offset value may also be changed any number of times while the motor is in motion. The changes that are made should be small; this will prevent sudden jumps in the motor motion.

10.Step Motor Control

10

In This Chapter

- Selecting the Step Motor Position Command Source
- Step Motor Waveform Generation
- Encoder Feedback

Overall, the control features of Juno when used with a step motor are similar to that when used with a servo motor. The primary differences between servo motors and step motors is that there is no position/outer loop or velocity loop module used for step motors and that motor output waveform generation is specific to the two-phase format used with step motors.

10.1 Selecting the Step Motor Position Command Source

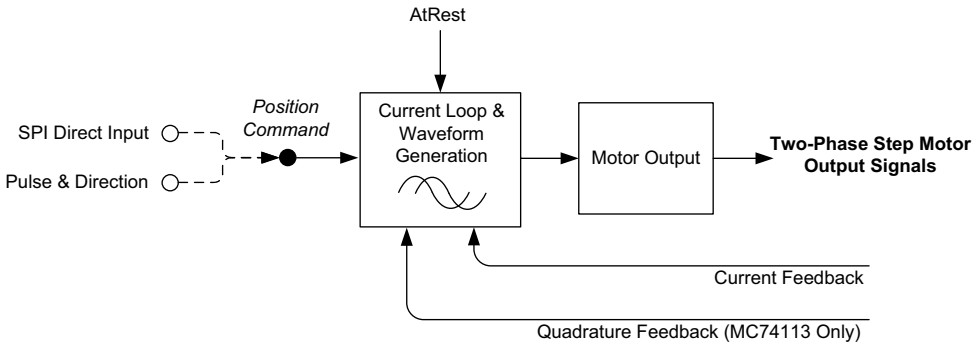


Figure 10-1: Step Motor Control Flow

Figure 10-1 provides an overview of the Juno IC control flow when driving step motors. Two position command sources are available; pulse & direction signal inputs and the direct input SPI port.

10.1.1 Pulse & Direction Command Input

When pulse & direction is selected as the command source the incoming *Pulse* & *Direction* signals are processed by the Juno and the resultant count is stored in a 32 bit position register. Pulse & direction input is a popular and easy to use method to continuously stream a commanded position to the Juno IC.

To select the step motor position command source the **SetDriveCommandMode** command is used. To read this value back the **GetDriveCommandMode** command is used. To read the current value of the command position register the **GetCommandedPosition** command is used.

For more information on *Pulse & Direction* signal input, timing, and related electrical information refer to the *MC78113 Electrical Specifications*.

10.1.2 Direct-Input SPI Command

Direct-input SPI is a convenient position command format for applications that generate the position profile via a microprocessor located on the same PCB. This is because most microprocessors provide one or more SPI ports however relatively few provide pulse & direction signal generation.

When direct-input SPI is selected the user continuously streams a signed 16-bit word containing the relative position change from the previous SPI direct input command. As for pulse & direction signal input, these direct input SPI commands are continuously processed and the resultant count is stored in a 32 bit position register.

To avoid jitter in the step motor motion the SPI direct input commands should be streamed at a fixed regular interval, varying in arrival time by no more than 1% of the interval between successive updates. For best performance the update frequency should be between 1.0 kHz and 10 kHz.

For additional information on direct input SPI formats refer to Section 14.3, "Direct Input SPI (Serial Peripheral Interface)." For detailed electrical information on the SPI bus refer to the *MC78113 Electrical Specifications*.

10.2 Step Motor Waveform Generation

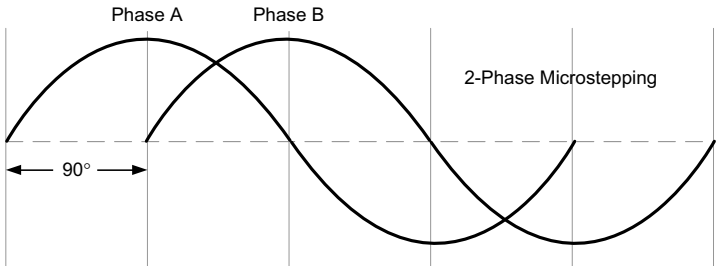


Figure 10-2: Microstepping Waveforms

When driving step motors Juno generates sinusoidal waveforms consisting of phase A and phase B outputs separated by 90 degrees. This is shown in Figure 10-2. The overall sinusoidal waveform is broken into discrete microstep positions, with the number of microsteps per full step (a full step is one quarter of a full electrical cycle) being user settable. Each microstep represents a discrete addressable position of the step motor, and thus the greater the number of microsteps per full step the greater the number of resolvable motor positions.

The number of microsteps per full step is specified using the phase counts register of the **SetCommutationParameter** command. The phase counts setting represents the number of microsteps per electrical cycle (four times the desired number of microsteps per full step).

For example if a 1.8° step motor (1.8° of motor rotation per full step) is set up with a resolution of 64 microsteps per full step the phase counts setting will be 64 x 4 = 256, and the number of resolvable positions per mechanical rotation will be 360° / 1.8° x 64 = 12,800 microsteps per motor rotation. Note that this represents the theoretical positioning resolution, but does not necessarily mean the motor will have this precision or even this number of actual resolved mechanical positions. This depends on a number of application characteristics including system friction, drive, torque, and motor linearity.

The minimum number of microsteps per full step is one (phase counts setting of four). The maximum number of microsteps per full step is 256 (phase counts setting of 1,024).

10.2.1 DriveCurrent

A user-specified drive current controls the amplitude of the microstepping waveform. To set the drive current the command **SetCurrent** is used. A value between 0 and 32,767 is set, representing an amplitude of zero to 100 percent.

Depending on whether the current loop module is enabled the provided drive current value will express either a current or a voltage. With the current loop active a drive current command of, for example, 100 % will command the highest possible available current to the motor. With the current loop not active (Juno operating in voltage mode) a 100% drive current command will command the highest possible available voltage.

For information on converting commanded current values to actual delivered amps at the motor refer to Section 14.1.1, "CurrentA-D Signals."

Determining what drive current to specify when the step motor is actively moving is generally done via trial and error. Higher values allow greater acceleration and top speed but generate more heat in the motor.

10.2.2 Holding Current and the AtRest Signal

Juno ICs provide an *AtRest* signal input which is used to indicate whether the axis is in motion. In conjunction with this signal a holding current command can be defined by the user which is only applied when the motor is at rest (not moving). By applying the holding current when the motor is not moving heat generation can be significantly reduced.

The holding current value is set using the command **SetCurrent**. To read the value set use the command **GetCurrent**. Note that the value specified actually represents the limit of the output current while the motor is in a holding condition. For example if the drive current is lower than the holding current, the drive current value rather than the holding current will be used.

A bit indicating whether the axis is currently in a holding condition is available in the Drive Status register. To read this register use the command **GetDriveStatus**.

10.3 Encoder Feedback

The Juno 74113 step motor control IC supports encoder input while the MC75113 step motor control IC does not. See Section 8.1.2, "Quadrature Encoder Input" for more information on encoder position feedback.

Most encoder commands operate as for servo motors. For example the current position is retrieved using the command **GetActualPosition**, the position capture location is retrieved using **GetCaptureValue**, and the **AdjustActualPosition** and **SetActualPosition** commands may be used to alter the current position.

10.3.1 Encoder to Microstep Ratio

In many step motor systems, the ratio of microsteps to encoder counts is not necessarily exactly one. Juno accommodates this by allowing the ratio of encoder counts to microsteps to be explicitly specified using the command **SetEncoderToStepRatio**. This value can be read back using the command **GetEncoderToStepRatio**.

This command accepts two parameters: the first parameter is the number of encoder counts per motor rotation, and the second parameter is the number of microsteps per motor rotation.

For example, if a step motor with a 1.8 degree full step size is used with an encoder with 4,000 counts per motor rotation, the encoder and microsteps per rev parameters would be 4,000 and 200 (360/1.8) respectively.

In cases where the number of steps, microsteps, or encoder counts per rotation exceeds the allowed maximum of 32,767, the parameters may be specified as fractions of a rotation, as long as the ratio is accurately maintained.

10.3.2 Encoder Position Units

With the MC74113 Juno IC, or when a MC78113 is used with the motor type set to step motor, the actual position units are in microsteps by default. If desired the actual position units can be set to encoder counts using the command **SetActualPositionUnits**. The **SetActualPosition**, **GetActualPosition**, **AdjustActualPosition** and **GetCaptureValue** commands are affected by this setting.

If the actual position units are set to encoder counts, then actual position comes directly from the encoder input. If the units are set to microsteps, then the encoder input is converted to microsteps using the value specified by **SetEncodertoStepRatio** command.

10.3.3 Stall Detection

In addition to passively returning the measured motor position via the **GetActualPosition** command, Juno ICs can actively monitor the encoder position and detect a motion error. The motion error mechanism allows Juno to detect when the step motor has stalled or otherwise lost steps during motion. This typically occurs when the motor encounters an obstruction, or otherwise exceeds its rated torque specification.

Automatic stall detection operates continuously once it is initiated. The current desired position (commanded position) is compared with the actual position (from the encoder), and if the difference between these two values exceeds a specified limit, a stall condition is detected. The user-programmed register **SetLoop** determines the threshold at which a motion error is generated.

Processing of a motion error while using a step motor is identical to that for servo motors. See Section 3.5, "Motion Error Detection" for details.

11. Amplifier Safety & DC Bus Monitoring

In This Chapter

Juno supports automatic detection of major amplifier, voltage supply, or other electrical hardware problems. The following sections describe these safety features.

Figure 11-1 provides an overview of amplifier & DC safety-related signals. For more information on interfacing to these signals as well as information on signal scaling and signal conditioning circuitry refer to the *MC78113 Electrical Specifications*.

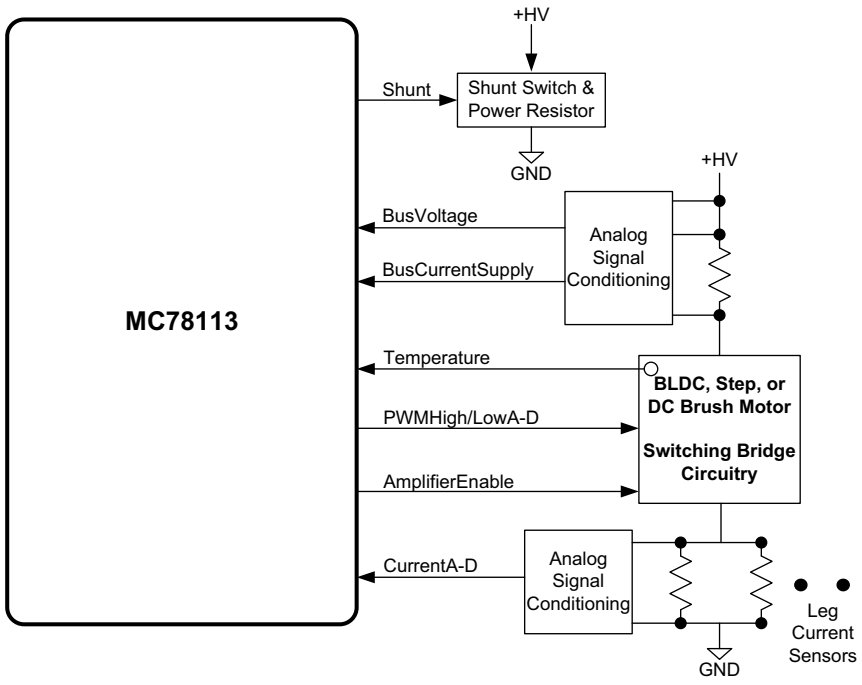


Figure 11-1: Amplifier & DC Bus Connections

11.1 Overtemperature Sense

Juno provides the capability to continually monitor temperature using a direct input sensor. Usually the temperature sensor is placed on or near the switching amplifier circuitry, but ultimately this is up to the user. A programmable value

set using the command **SetDriveFaultParameter** is compared to a value read from Juno's *Temperature* signal, and if the value read from the sensor exceeds the programmed threshold, an overtemperature fault occurs. Both temperature increasing/voltage increasing and temperature increasing/voltage decreasing sensors can be accommodated. To read the value set the command **GetDriveFaultParameter** is used.

An overtemperature fault will cause the following events:

- Depending on how event processing has been set one or more modules may be disabled.
- The overtemperature bit in the Event Status register is set active.

To recover from this condition, the user should determine the reason for the fault and correct accordingly. It is always the responsibility of the user to maintain safe operating conditions of the drive and associated electronics. Once this has occurred, the overtemperature bit of the Event Status register should be cleared. This can be accomplished using **ResetEventStatus**. The normal operation of the control modules can then be restored using **RestoreOperatingMode**.

The instantaneous status of the overtemperature threshold comparison can be read using the command **GetDriveStatus**. If the overtemperature fault condition is still occurring at the time the overtemperature bit of the Event Status register is cleared, this bit will immediately be set again, and the recovery sequence must be executed again.

Once programmed the temperature comparison function operates continuously. To disable it, a value of 32,767 should be programmed. To read the current value of the temperature sensor, the command **GetTemperature** is used.

11.2 Overvoltage Sense

Juno provides the capability to sense overvoltage conditions in the main +HV bus voltage. A programmable threshold set using the command **SetDriveFaultParameter** is compared to the value read from the drive DC bus supply via the *BusVoltage* signal input, and if the value read exceeds the programmed threshold, an overvoltage fault occurs. To read the value set, the command **GetDriveFaultParameter** is used.

An overvoltage fault will cause the following events:

- Depending on how event processing has been set one or more modules may be disabled.
- The drive exception bit in the Event Status register becomes active.
- The overvoltage bit in the Drive Fault Status register is set.

To recover from this condition, the user should determine the reason for the fault and correct accordingly. It is always the responsibility of the user to maintain safe operating conditions for the drive electronics. Once this has occurred, the overvoltage bit of the Drive Fault Status register should be cleared using the **ClearDriveFaultStatus** command and the drive exception bit of the Event Status register should be cleared using the **ResetEventStatus** command. The normal operation of the control modules can then be restored using **RestoreOperatingMode**.

The instantaneous value of the overvoltage condition can be read using the command **GetDriveStatus**. If the overvoltage fault condition is still occurring while the overvoltage bit of the Event Status register is being cleared, this bit will immediately be set again, and the recovery sequence described above must be executed again.

Once programmed the temperature comparison function operates continuously. To disable it, a value of 32,767 should be programmed. The drive supply voltage can be monitored using the **GetBusVoltage** command. It returns the current supply voltage reading.

11.3 Undervoltage Sense

Juno provides a capability very similar to the overvoltage sense except that it monitors undervoltage. To set the programmable threshold the command **SetDriveFaultStatus** is used. This value is compared to the value read from the *BusVoltage* signal, and if the value read is less than the programmed threshold, an undervoltage fault occurs. To read the value set, the command **GetDriveFaultStatus** is used.

Threshold units, recovery procedure, and all other aspects of this feature are the same as for overvoltage sense except that the bit status location in the Drive Fault Status register is different. And just as for overvoltage conditions, it is the user's responsibility to determine the seriousness of, and appropriate response to, an undervoltage condition.

11.4 Overcurrent Sense

Juno provides the capability to sense overcurrent conditions in both the supply and the return of the DC Bus. A programmable threshold set using the command **SetDriveFaultParameter** is compared to the value read from the DC Bus current supply via the *BusCurrentSupply* signal input, and if the value read exceeds the programmed threshold, an overcurrent fault occurs. To read the value set, the command **GetDriveFaultParameter** is used.

The DC Bus return current is determined from Juno's leg current sensor inputs. To set this comparison threshold the **SetDriveFaultParameter** command is used and this value can be read back using **GetDriveFaultParameter**.

An overcurrent fault will cause the following events:

- Depending on how event processing has been set one or more modules may be disabled.
- The drive exception sense bit in the Event Status register becomes active

To recover from this condition, the user should determine the reason for the fault and correct accordingly. It is always the responsibility of the user to maintain safe operating conditions for the drive electronics. Once this has occurred, the DriveException bit of the Event Status register should be cleared. This can be accomplished using **ResetEventStatus**. The normal operation of the control modules can then be restored using **RestoreOperatingMode**.

If the overcurrent fault condition is still occurring while the DriveException bit of the Event Status register is being cleared, this bit will immediately be set again, and the recovery sequence described above must be executed again.

Once programmed the temperature comparison function operates continuously. To disable it, a value of 32,767 should be programmed.

11.5 Drive Enable

Juno supports an *Enable* input signal that must be active for proper operation. This signal is useful for allowing external hardware to indicate a fault to Juno and thereby automatically shut it down. The signal has an active low interpretation which can not be changed by the **SetSignalSense** command.

If the *Enable* signal becomes inactive (goes high) the following events occur:

- Depending on how event processing has been set one or more modules may be disabled.
- The disable bit in the Event Status register becomes active.

To recover from this condition the user should determine the reason for the enable becoming inactive, and correct accordingly. Once this has occurred, the appropriate bit of the Event Status register should be cleared. This can be

accomplished using **ResetEventStatus**. The normal operation of the control modules can then be restored using **RestoreOperatingMode**.

If the *Enable* signal is still inactive while the disable or drive exception bit of the Event Status register is being cleared, this bit will immediately be set again, and the recovery sequence must be executed again.

The status of the *Enable* signal can be read using the command **GetSignalStatus**.

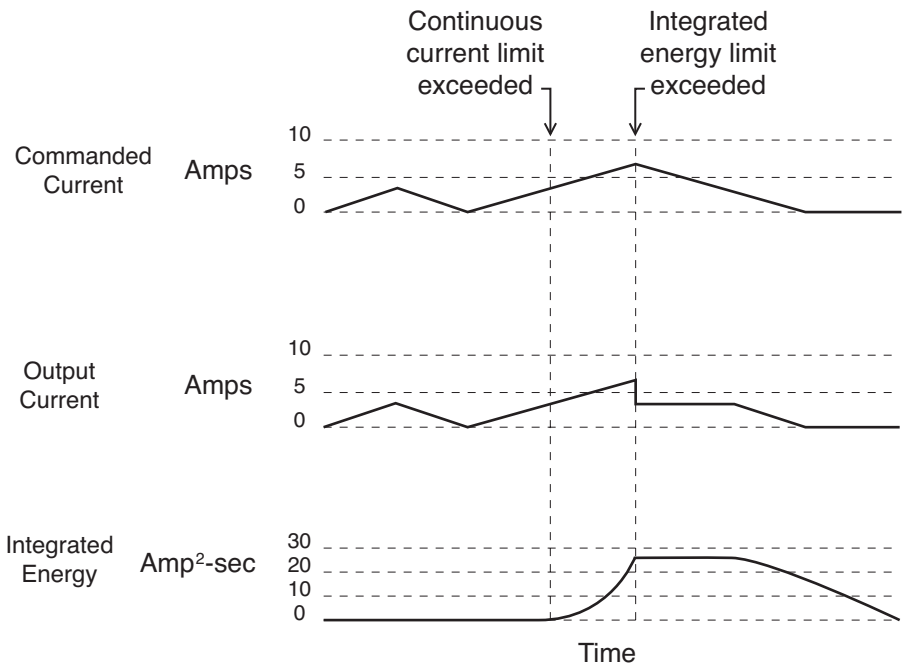
11.6 Current Foldback

Juno supports a current foldback feature, sometimes referred to as an I^2t foldback, which can be used to protect the drive output stage or motor windings from excessive current. I^2t current foldback works by integrating, over time, the difference of the square of the actual motor current and the square of the user-settable continuous current limit.

When the integrated value reaches a user-settable energy limit, Juno goes into current foldback. The default response to this event is to cause the current loop and motor output modules to be disabled. However it is also possible to program Juno to attempt to clamp the maximum current to the continuous current limit value. Note that Juno's ability to do so depends on a properly functioning current loop.

Juno will stay in foldback until the integrator returns to zero. This is shown in Figure 11-2.

Figure 11-2: Current Foldback Processing Example



Setting continuous current limit and energy limit to less than the maximum available from the amplifier circuitry is useful if the required current limit is due to the motor, rather than to the drive electronics. Continuous Current Limit and Energy Limit can be set using the command **SetCurrentFoldback**. The values set using this command can be read back using **GetCurrentFoldback**.

To disable current foldback operation, which is also the default condition, one or both of these parameters should be set to 32,768.

The instantaneous state of current foldback (whether the foldback limit is active or not) is available in the Drive Status register and can be read using the command **GetDriveStatus**. In addition, if a foldback event has occurred, this event is recorded in the Event Status register and can be read back using **GetEventStatus**.

Example

A particular motor has an allowed continuous current rating of 3 amps. In addition, this motor can sustain a temporary current of 5 amps for 2 seconds.

In this example the *continuous current limit* would be set to 3 amps, and the energy limit would be set to:

Energy Limit = (peak current^2 - continuous current limit^2) * time
Energy Limit = (5^2A^2 - 3^2A^2) * 2 Sec
Energy Limit = $32A^2$Sec

Current foldback, when it occurs, may indicate a serious condition affecting motion stability, smoothness, and performance. It is the responsibility of the user to determine the appropriate response to a current foldback event.

11.6.1 Current Foldback in Voltage Mode

Current foldback still operates when Juno is in voltage mode (current loop disabled). When in this mode, the I^2t energy calculations and condition testing are identical as when Juno is operating in current control mode.

Nevertheless, when in voltage mode, there is an important operational difference. In particular, if the limit is exceeded, rather than clamping the maximum current output to the programmable maximum continuous current limit setting, the power stage module is disabled or enters a braking state, thereby halting further motor output.

11.7 Shunt Signal

As shown in Figure 11-1 Juno can control a shunt PWM output, which in turn typically drives a MOSFET or IGBT switch which connects the HV bus voltage to the DC bus ground via a power resistor, thereby lowering the DC bus voltage. The shunt function can be used with DC Brush or Brushless DC motors but is typically not used with step motors. The shunt is useful for controlling excessive DC bus voltage particularly if the servo motor is decelerating rapidly resulting in back-EMF generation.

The shunt functions by continually comparing the DC bus voltage, as presented at the ***BusVoltage*** signal, to a user programmable threshold. If the DC bus voltage exceeds the comparison threshold the ***Shunt*** signal outputs a PWM waveform at a user programmable duty cycle. This PWM frequency is equal to the motor drive PWM frequency. Once active, shunt PWM output will stop when the DC bus drops to 2.5% below the threshold comparison value.

Once programmed, the shunt comparison function operates continuously. To disable it, a value of 32,767 should be programmed. The shunt function is not active when motor output is not enabled (the active operating mode output bit is not set). Similar to the motor amplifier PWM outputs, the signal sense of the shunt PWM output signal can be user programmed using the **SetPWMDrive** command. The default value is active high.

For additional information on the shunt function refer to the *MC78113 Electrical Specifications.*

This page intentionally left blank.

12.Power-up, Configuration Storage & NVRAM

In This Chapter

- Power-up
- Initialization Execution Control
- Initialization Monitoring
- Non-Volatile (NVRAM) Storage
- NVRAM Setup in the Production Application

12.1 Power-up

After receiving stable power at the Vcc pins Juno begins its initialization sequence. In a power-up where no user-provided configuration settings have been stored this takes approximately 250 mSec. At the end of this sequence all parameters are at their default values, and both the current loop module and the power stage module are disabled. At this point Juno is ready to receive host commands and begin operation.

Juno also supports the ability to store configuration settings that are applied during the power up sequence. For this purpose, Juno supports a 1,024 word memory that is non-volatile (NVRAM), meaning the data stored will be available even after power to the Juno IC is removed.

The power-up initialization information stored in the NVRAM takes the form of host packets, however rather than being sent via a host communication port, these packets are stored in memory. If the non-volatile memory has been loaded with configuration information the power-up sequence detects this and begins executing these commands. Note that processing stored commands may increase the overall initialization time depending on the command sequence stored.

12.2 Initialization Execution Control

To make the initialization sequence as flexible as possible Juno provides a facility to control execution of the initialization commands stored in NVRAM. Command execution can be suspended for a specific period of time, or until various internal or external conditions are satisfied. This is useful for coordinating Juno IC startup with external processes on the user's controller board, to synchronize completion of Juno motor initializing sequences such as pulse phase initialization, or to execute simple motion sequences prior to normal operation.

This facility is executed via the **ExecutionControl** command. After calling this command further initialization command processing is suspended until the specified execution control condition is satisfied or until a user-specified timeout interval elapses.

12.2.1 Settable Parameters

The following table indicates the parameters that are specified with the **ExecutionControl** command:

Parameter	Description
Condition	This field selects the execution control condition. The options are delay a specified amount of time, compare against the Event Status register, compare against the Activity Status register, compare against the Drive Status register, or compare against the Signal Status register.
Delay time	This field is specified if the condition is set to delay. The delay time is a 32-bit number that holds the number of cycle times to delay.
Compare masks	This field is specified if a compare condition is selected. Two 16-bit numbers are specified representing a compare mask and a compare sense value.
Timeout	Specifies the maximum amount of time to wait for the condition to become true. A value of zero means never time out. If the condition times out an error is raised, the motor output module is disabled, and initialization halts.

Refer to the *Juno Velocity & Torque Control IC Programming Reference* for the exact encoding of the **ExecutionControl** command.

12.2.1.1 Compare Masks

Juno allows a broad range of internal or external conditions to be used for execution control during initialization. This allows initialization command sequences such as "Start the pulse phase initialization and wait till it completes before continuing," "Rotate the motor after *HallA* signal goes high," and "Change the profile target velocity once the profile velocity has reached zero."

These execution control comparisons are specified via a 16-bit mask value and a 16-bit sense value. The 16-bit mask value contains a 1 in each bit position of the corresponding specified status register that will be compared against, and a 0 for bits that will not be compared. The 16-bit sense value contains the value for each bit of the status register that will be compared.

Example

A user wants to suspend execution for up to 30 seconds during initialization until the *Index* and *AtRest* pin signals are high (1) and low (0) respectively. Since these signal states are held in the Signal Status register at bit positions 2 and 7 the user sets the execution condition for comparison of the Signal Status register, a compare mask value of $2^2 + 2^7 = 36$ and a compare value would be $2^2 + 0^7 = 4$. The user also specifies a timeout value of 30 seconds.

12.3 Initialization Monitoring

To determine if initialization is complete a flag indicating this in the Drive Status register is set. Refer to Section 8.2.2, "Activity Status Register" for more information. For more information on how the initialization commands are stored into NVRAM see Section 12.4, "Non-Volatile (NVRAM) Storage." For detailed electrical information on Juno reset and powerup refer to the *MC78113 Electrical Specification*s.

If there are errors in the stored command sequence then an instruction error will be set so that the error can later be diagnosed. Juno will abort initialization if it detects any error while processing commands.

The host controller may poll the Drive Status register to determine when initialization is complete. If an error is detected the host controller can send a **GetInstructionError** to diagnose the nature of the erroneous command processed during initialization.

12.4 Non-Volatile (NVRAM) Storage

A primary purpose of the NVRAM is to allow Juno initialization information to be stored so that upon power up it can be automatically loaded rather than requiring an external controller to perform this function. In addition the NVRAM can be used for other functions such as labeling the stored initialization sequence, or for general purpose user-defined storage.

All data stored in the Juno NVRAM utilizes a data format known as PMD Structured data Format (PSF). Users who rely only on PMD's Pro-Motion software package to communicate with Juno and store and retrieve initialization parameters need not concern themselves with the details of PSF. Users who want to address the NVRAM from their own software, or who want to create their own user-defined storage on the Juno NVRAM will require the PSF format details, which are documented in the *Juno Velocity & Torque Control IC Programming Reference*.

The following section describes how writing and reading to the NVRAM memory space is accomplished.

12.4.1 Writing to NVRAM

There are significant restrictions to writing to Juno's NVRAM area. In particular it is not possible to erase and rewrite selected sections of the memory space. Only the sequence described in this section can be used to write memory into the user NVRAM space.

Data stored into the NVRAM area must follow PSF (PMD Structured data Format). Failure to do so may result in unexpected behavior of the Juno.

The following sequence is used to store command initialization data or other data to the non-volatile memory area:

1. Send a **NVRAM** command with an argument of *NVRAMMode*. Sending this command places Juno in a special mode allowing it to store memory into the NVRAM. Before proceeding the host controller should delay 1 second or more.
2. Send a **NVRAM** command with an argument of *EraseNVRAM*. This command will erase the entire NVRAM memory area. Before proceeding the host controller should delay four seconds or more.
3. For each 16-bit word of data that is to be written into the NVRAM area the command **NVRAM** with an argument of *Write* is sent, along with the data word to be written. After each word is written Juno increments an internal pointer so that subsequent data words are automatically stored in the correct location.
4. Once all data is successfully written the host controller should send a **Reset** command, which will cause Juno to reboot and execute a power up sequence. Note that this power-up sequence will include processing the stored data sent using the above sequence.

If an error occurs when processing NVRAM the instruction error event bit will be set and the **GetInstructionError** command may be used to read the error code.

It is not possible to write to the NVRAM area using the buffer commands. The procedure outlined in Section 12.4.1, "Writing to NVRAM" must be used to write data to the NVRAM area.

12.4.2 Reading from NVRAM

If desired, it is possible to directly read the NVRAM memory area using buffer commands. See Section 8.5.7, "Buffer Commands" for more information on buffer processing.

For convenience the NVRAM buffer is pre-defined with an ID# of 1. The **ReadBuffer16** command can be used to read all or part of the NVRAM space.

12.5 NVRAM Setup in the Production Application

When a user programmable microprocessor is located on the user's control board the most common approach for configuring Juno for operation is for the microprocessor to send a sequence of host commands to Juno, thereby initializing and configuring it for operations.

If there is no host microprocessor however and Juno's direct input control modes are being used, Juno's NVRAM must be loaded with this command sequence so that it can initialize and configure itself automatically during powerup.

For more information refer to Section 2.9, "Juno ICs in the Production Application."

13.Host Communication

13

In This Chapter

- Host Communications
- Host Commands
- Serial Communications
- Controller Area Network (CAN)
- SPI (Serial Peripheral Interface) Communications

13.1 Host Communications

By communicating via commands the host can control the behavior of the motion system as desired while monitoring the status of the Juno IC and the motor. Juno ICs support several methods of host communication including point-to-point and multi-drop asynchronous serial, CANbus 2.0, and SPI (Serial Peripheral Interface). However not all Juno ICs support all host communications modes. This is shown in the table below:

Juno IC Type	P/Ns	Serial Point-to-point	Serial Multi-drop	CANbus	SPI*
Velocity control	MC71113, MC73113, MC78113	✓	✓	✓	✓
Torque control	MC71112, MC71112N, MC73112, MC73112N	✓			
Step motor control	MC74113, MC74113N, MC75113, MC75113N	✓			

**while Juno torque control and step motor control ICs do not support SPI for host communications, they do provide SPI input for command value input or for sensor reading input.*

13.2 Host Commands

All communications to and from the chipset, whether serial, CANbus, or SPI, are in the form of packets. A packet is a sequence of transfers to and from the host, resulting in a Juno action or data transfer. Packets may consist of a command with no data (dataless command), a command with associated data that are written to the chipset (write command), or a command with associated data that are read from the chipset (read command).

Every command sent by the host has a structured format that does not change, even if the amount of data and nature of the command vary. Each command has an instruction word (16 bits) that identifies the command. There may be zero or more words of data associated with the command that the host writes to the IC. This is followed by zero or more words of data that the host reads from the IC. The type of command determines whether there are data written to Juno and to the host.

Most commands with associated data (read or write) have one, two, or three words of data. See the *Juno Velocity & Torque Control IC Programmer's Reference* for more information on the length of specific commands. If a read or a write command

has two words of associated data (a 32-bit quantity), the high word is loaded/read first, and the low word is loaded/read second.

Design Note: While some users will develop their own low-level libraries for interfacing to a Juno IC, PMD's higher-level language C-Motion, provides convenient C-language APIs (Application Programmmer Interface) for all Juno commands.

13.3 Serial Communications

All Juno ICs provide an asynchronous serial connection. This serial port may be configured to operate at baud rates ranging from 1200 baud to 460,800 baud and may be used in point-to-point or multi-drop mode.

13.3.1 Configuration

The following table shows the Juno default serial settings:

Communications Mode	Default Value
point-to-point serial	57.6K, 1 stop bit, no parity.
multi-drop serial	57.6K, 1 stop bit, no parity, point-to-point mode.

The basic unit of serial data transfer (both transmit and receive) is the asynchronous frame. Each frame of data consists of the following components.

- One start bit.
- Eight data bits.
- An optional even/odd parity bit.
- One or two stop bits.This data frame format is shown in the following figure.

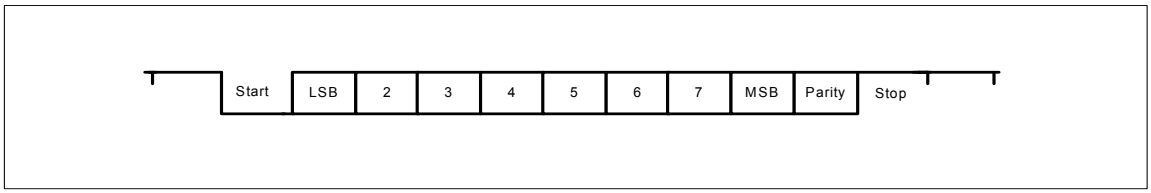


Figure 13-1: Typical Data Frame Format

13.3.2 Command Format

The command format that is used to communicate between the host and Juno consists of a command packet sent by the host processor, followed by a response packet sent by Juno. The host must wait for, receive, and decode the response packet.

Command packets sent by the host contain the following fields.

Field	Byte#	Description
Address	1	One byte identifying the Juno IC to which the command packet is being sent. This field should always be zero in point-to-point mode.
Checksum	2	One byte value used to validate packet data. See the table in Section 13.3.4, "Checksums" for detailed information.

Field	Byte#	Description
Instruction code	3–4	Two byte instruction, sent upper byte (axis number) first. The command codes are detailed in the *Juno Velocity & Torque Control IC Programming Reference*.
Data	5	Zero to six bytes of data, sent most significant byte (MSB) first. See the individual command descriptions for details on data required for each command.

In response to the command packet, Juno will respond with a packet in the following format.

Field	Byte#	Description
Address	(1)*	One byte identifying the Juno IC sending the response. Present in multi-drop mode only.
Status	1 (2)	Zero if the command was completed correctly; otherwise, an error code specifying the nature of the error. See Section 13.3.3, "Instruction Errors" for more information.
Checksum	2 (3)	A one-byte checksum value used to validate the packet's integrity. See details in the preceding table.
Data	3 (4)	Zero to six bytes of data. No data will be sent if an error occurred in the command (i.e. the status byte was non-zero). If no error occurred, then the number of bytes of data returned would depend on the command to which Juno was responding. Data are always sent MSB first.

* Note that the address byte is only present in the response packet when in multi-drop mode. In this case, its is also included in the checksum calculation.

13.3.3 Instruction Errors

There are a number of checks made by the Juno IC on commands it receives. These checks improve safety of the motion system by eliminating some obviously incorrect command data values. All such checks associated with host commands are referred to as instruction errors. The status byte in the response packet will contain one of the error codes.

13.3.4 Checksums

Both command and response packets contain a checksum byte. The checksum is used to detect transmission errors, and allows Juno to identify and reject packets that have been corrupted during transmission or were not properly formed.

Checksums are mandatory when using serial communications. Any command packets sent to Juno containing invalid checksums will not be processed and will result in a data packet being returned containing an error status code.

The serial checksum is calculated by summing all bytes in the packet (not including the checksum) and negating (i.e., taking the two's complement of) the result. The lower eight bits of this value are used as the checksum. To check for a valid checksum, all bytes of a packet should be summed (including the checksum byte), and if the lower eight bits of the result are zero, then the checksum is valid.

Example

If a command packet is sent to address 3, containing command 077h **SetMotorCommand** with the one-word data value 1234h, then the checksum will be calculated by summing all bytes of the command packet (03h + 77h + 12h + 34h = C0) and negating this to find the checksum value (40h). On receipt, Juno will sum all bytes of the packet, and if the lower eight bits of the result are zero, then it will accept the packet (03h + 040h + 77h + 12h + 34h = 100h).

13.3.5 Transmission Protocols

The Juno ICs support the ability to have more than one Juno IC on a serial bus, thereby allowing a chain, or network of ICs, to communicate on the same serial hardware signals.

There are two methods supported by the serial port to resolve timing problems, transmission conflicts, and other issues that may occur during serial operations. These are point-to-point (used when there is only one device connected to the serial port) and multi-drop idle-line mode (used when there are multiple devices on the serial bus). The following sections describe these transmission protocols.

13.3.6 Point-to-Point Mode

Point-to-point serial mode is intended to be used when there is a direct serial connection between one host and one Juno IC. In this mode, the address byte in the command packet is not used by Juno IC (except in the calculation of the checksum), and Juno responds to all commands sent by the host.

When in point-to-point mode, there are no timing requirements on the data transmitted within a packet. The amount of data contained in a command packet is determined by the command code in the packet. Each command code has a specific amount of data associated with it. When Juno receives a command code, it waits for all data bytes to be received before processing the command. The amount of data returned from any command is also determined by the command code. After processing a command, Juno will respond with a data packet of the necessary length. No address byte is sent with this response packet.

When running in point-to-point mode, there is no direct way for Juno to determine the beginning of a new command packet, except by context. Therefore, it is important for the host to remain synchronized with Juno when sending and receiving data. To ensure that the processors remain synchronized, it is recommended that the host processor implement a time limit when waiting for data packets to be sent by Juno. The suggested minimum timeout period is the amount of time required to send one byte at the selected baud rate plus one millisecond. For example, at 9600 baud each bit takes 1/9600 seconds to transfer, and a typical byte consists of 8 data bits, 1 start bit, and 1 stop bit. Therefore, one byte takes just over 1 millisecond, and the recommended minimum timeout is 2 milliseconds.

If the timeout period elapses between bytes of received data while the host is waiting on a data packet, then the host should assume that it is out of synchronization with Juno. To resynchronize, the host should send a byte containing zero data and wait for a data packet to be received. This process should be repeated until a data packet is received from Juno; at which point the two processors will be synchronized.

13.3.7 Multi-drop Idle-line Mode

This multi-drop protocol is intended to be used on a serial bus in which a single host processor communicates with multiple Juno ICs (or other subordinate devices). In this mode, the address byte that starts a command packet is used to indicate the device for which the packet is intended. Only the addressed device will respond to the packet.

When the idle-line mode is used, Juno imposes tight timing requirements on the data sent as part of a command packet. In this mode, Juno will interpret the first byte received after an idle period as the start of a new packet. Any data already received will be discarded.

The timeout period is equal to the time required to send ten bits of serial data at the configured baud rate—for example, roughly 1 millisecond at 9600 baud. If a delay of this length occurs between bytes of a command packet, then the bytes already received will be discarded, and the first character received after the delay will be interpreted as the address byte of a new packet.

Once Juno receives an address byte and a command code, it waits for all data bytes to be received before processing the command. The amount of data returned from any command is also determined by the command code. After processing a command, Juno will respond with a data packet of the necessary length. In multi-drop mode, the first byte of every response packet contains the address of the responding Juno IC. This prevents other devices on the network from interpreting the response as a command sent to them.

Note that this multi-drop protocol may also be used when the host and Juno IC are wired in a point-to-point configuration, as long as the host always transmits the correct address byte at the start of a packet and follows any

additional rules for the selected protocol. This mode of operation allows the host to ensure that it will remain synchronized with the Juno IC without implementing the timeout and re-synch procedure previously outlined.

13.4 Controller Area Network (CAN)

CAN is a serial bus system especially suited for networking "intelligent" devices as well as sensors and actuators within a system or sub-system.

13.4.1 Overview

All Juno ICs provide a CAN 2.0B network and will coexist (but not communicate) with CANopen nodes on that network. Juno uses CAN to receive commands, send responses, and (optionally) send asynchronous event notifications. Each message type has an address, as shown in the following table.

Message Type	CAN Address
Command received	0x600 + nodeID
Command response	0x580 + nodeID
Event notification	0x180 + nodeID

CAN nodes communicate via messages. Each message may carry a data payload of up to 8 bytes. The CAN interface layer automatically corrects transmission errors. Unlike the serial protocols, a checksum is not a part of the Juno's CAN interface protocol.

13.4.2 Message Format

Messages are transmitted and received using the standard format identifier length of 11 bits. All network messages that use the extended format 29-bit identifier are ignored by Juno. Commands have varying data lengths; see the *Juno Velocity & Torque Control IC Programming Reference* for the data formats of particular commands. Correspondingly, in the following table, not all of the data word bytes will always be present depending on the command being processed.

The corresponding byte sequences in the CAN protocol for the three message types are described in the following tables.

Command Received	
Message data byte	**Corresponding parallel byte**
1	Command word, high byte
2	Command word, low byte
3	1st data word, high byte
4	1st data word, low byte
5	2nd data word, high byte
6	2nd data word, low byte
7	3rd data word, high byte
8	3rd data word, low byte

Command Response	
Message data byte	**Corresponding parallel byte**
1	*Reserved (always zero)*

Command Response	
Message data byte	**Corresponding parallel byte**
2	Instruction status code
3	1st data word, high byte
4	1st data word, low byte
5	2nd data word, high byte
6	2nd data word, low byte
7	3rd data word, high byte
8	3rd data word, low byte

The first word in a response will contain a value of zero in the upper byte, and the lower byte will contain a value that will also be zero in a no-error condition, but will be non-zero if an error occurred while processing the instruction. (See Section 13.3.3, "Instruction Errors" for more information.) The byte following the status byte will be the high byte of the 1st data word, followed by the low byte of the 1st data word and continuing as shown in the preceding table. The actual number of bytes returned is determined by the instruction that was issued; see the *Juno Velocity & Torque Control IC Programming Reference* for the data lengths and formats of each command.

Event Notification	
Message data byte	**Data Interpretation**
1	zero
2	zero
3	Event Status register value, high byte
4	Event Status register value, low byte

The first and second bytes in a notification message will contain a value of zero. The 3rd and 4th bytes are the high and low byte of the Event Status register from the notifying axis.

13.4.3 Configuring the CAN Interface

Juno CAN interface may be configured via the command **SetCANMode**. This command is used to set the CAN nodeID of a particular Juno IC (0–127), as well as the transmission rate of the connected CAN network. The supported transmission rates are as follows:

SetCanMode Encoding	CAN Transmission Rate (bps)
0	1,000,000
1	*reserved*
2	500,000
3	250,000
4	125,000
5	50,000
6	20,000
7	10,000

13.4.4 CAN Event Notification

When communicating via the CAN interface, Juno may (optionally) send messages when selected bits in the Event Status register are set active. (See Section 8.2.1, "Event Status Register" for more information.) These messages are sent with a CAN address of **0x180 + nodeID**.

This CAN notification facility is controlled with the command **SetInterruptMask**. For each **on** bit in the notify mask, a CAN message will be generated whenever the corresponding bit in the Event Status register becomes 1.

13.5 SPI (Serial Peripheral Interface) Communications

Juno ICs support an SPI interface for communication with a host microprocessor or other controller. Note that this interface is intended for on-board interconnections only. SPI should not be used for off-board communication.

The Juno SPI interface utilizes three digital input pins *HostSPIEnable*, *HostSPIClock* and *HostSPIRcv*; and two digital output pins *HostSPIXmt*, and *HostSPIStatus*. These signals represent the standard SPI enable, chip select, clock, and data functions, along with a protocol packet processing status indicator. When used with SPI host communications Juno acts as an SPI slave, and the host processor acts as an SPI master.

The SPI port may also be used for direct input of torque, velocity, or outerloop quantities. For informaiton on operating the SPI port in direct input mode refer to Section 14.3, "Direct Input SPI (Serial Peripheral Interface).".

13.5.1 SPI Protocol, Command Send

Each overall host SPI data exchange consists of the host asserting *HostSPIEnable*, the host sending two or more 16-bit data words, and the host de-asserting *HostSPIEnable*.

SPI commands that do not include data to send to Juno (dataless commands) consist of two transmitted 16-bit words, commands that include one word of data consist of three transmitted 16 bit words, etc.

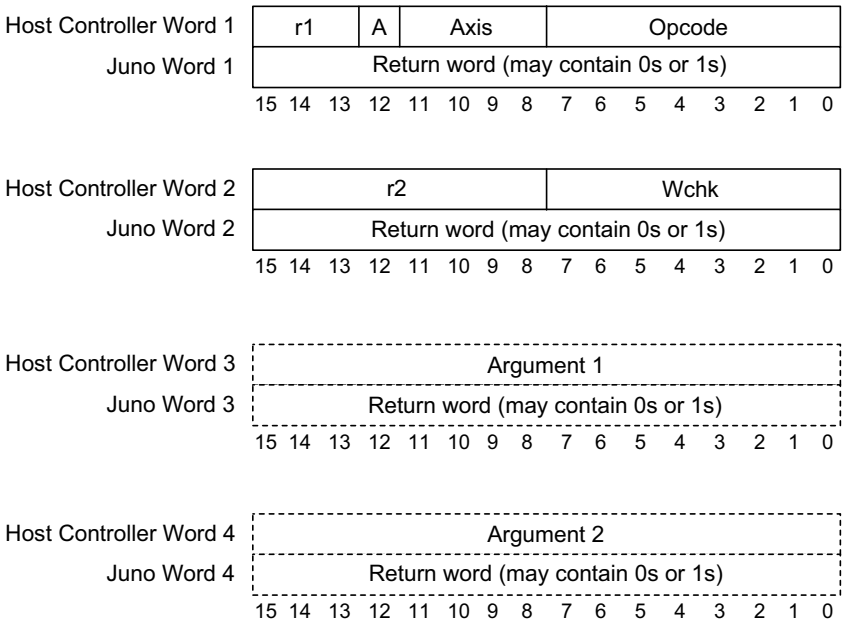


Figure 13-2: SPI Command Send Packet Sequence

This is shown in Figure 13-2 which shows the overall packets sequence and format for SPI command write communications. The following table details the content of these packets:

Field	Bit	Name	Description

Opcode	0-7	Opcode	Contains the 8 bit command opcode
r1	8-15	*Reserved*	*Reserved, must contain 0.*
Wchk	0-7	Write checksum	Contains the logical negation of an 8-bit ones complement checksum computed over all bits in the command packet except for the checksum field, and a seed of 0xAA. If the checksum received by Juno is incorrect (does not equal 0xff), the command will not be executed and will returns a checksum error code during the next data exchange.
r2	8-15	*Reserved*	*Reserved, must contain 0.*

The additional word writes argument1, argument2, shown in Figure 13-2 contain data (if any) associated with the command being sent to Juno. For example if the command SetMotorCommand is being sent, then a single 16-bit data word, consisting of the programmed command value is transmitted in argument1. Only the required number of argument data words should be sent.

13.5.2 SPI Protocol, Command Response

After receiving a command with a valid checksum and the appropriate number of argument words Juno will process the command. After command processing is complete, Juno will drive the *HostSPIStatus* signal high. The host reads this signal state change, and then initiates an SPI exchange to retrieve the command status as well as any return data words that may have been associated with the command that was sent by the host.

Note that the host should begin polling *HostSPIStatus* only after de-asserting *HostSPIEnable*. Polling earlier may result in incorrect interpretation of the *HostSPIStatus* signal.

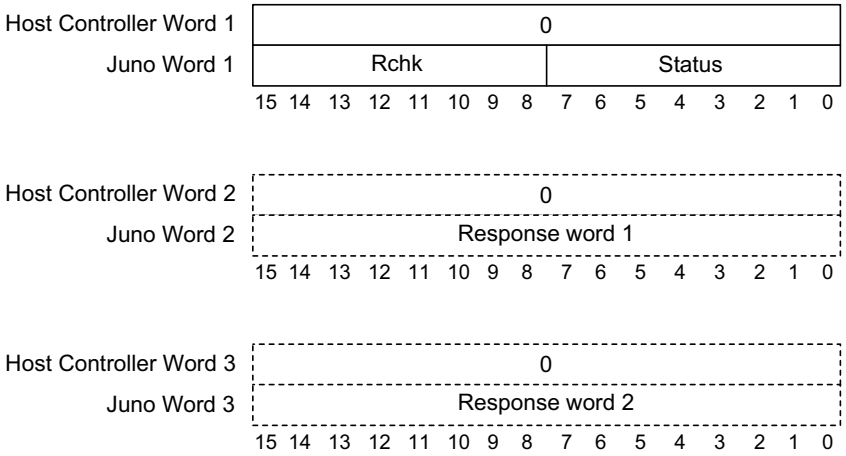


Figure 13-3: SPI Response Packet Sequence

Figure 13-3 shows the packet format to receive a command response. In all cases the host sends 16-bit data words consisting of zeroes, while simultaneously retrieving data from Juno. The following table describes the fields of this returned data:

Field	Bit	Name	Description
Status	0-7	Command Status	This fields contains an 8-bit word which provides a status of the processed command. Positive values indicate that command processing occurred normally with the returned value holding the number of 16-bit words to be returned. Negative values contain an error code, and indicate that command processing did not occur normally.
Rchk	8-15	Response checksum	Contains the logical negation of an 8-bit ones complement checksum computed over all bits of the return packet except the checksum, and a seed of 0xAA. It is not required, although highly recommended, that the host confirm that the return checksum is correct (totals to 0xFF).

The additional word reads response1, response2, shown in Figure 13-3 contain data (if any) associated with the command that was sent to Juno. For example if the command **GetMotorCommand** was sent, then a single 16-bit data word, consisting of the current value of Juno's motor command register, will be returned.

If the host attempts to retrieve command status before it is ready an error will occur. The signal *HostSPIStatus* should always be used to synchronize command status checks.

Even if no data is being returned by Juno, the host must retrieve the command response. Failure to do so will result in SPI communications becoming desynchronized.

13.5.3 Configuring Host SPI

There are no user-configurable parameters that need to be set for the Host SPI interface. For more information on host SPI interface timing and signalling refer to the *MC78113 Electrical Specifications*.

This page intentionally left blank.

14. Hardware Signals

In This Chapter

- Analog Signal Input
- Analog Signal Calibration
- Direct Input SPI (Serial Peripheral Interface)

14.1 Analog Signal Input

Juno provides direct analog signal input in conjunction with a number of its control related features. Analog signals are input in a voltage range of 0.0V to 3.3V and then converted via an internal ADC (analog to digital converter) into a digital word.

Analog signal input values can be read via the ReadAnalog command by specifying the analog channel that is being read. The following table summarizes the Juno analog inputs and the corresponding analog channel:

Juno Signal	Channel #	Description
CurrentA	0	Switching amplifier A leg current feedback input
CurrentB	1	Switching amplifier B leg current feedback input
CurrentC	2	Switching amplifier C leg current feedback input
CurrentD	3	Switching amplifier D leg current feedback input
Temperature	4	Temperature sensor input
BusCurrentSupply	5	DC Bus supply side current input
BusVoltage	6	DC Bus voltage input
AnalogCmd	7	Direct analog velocity or torque command input
Tachometer	8	Tachometer velocity or measured outer loop value input

The *MC78113 Electrical Specifications* has detailed information on how to connect to Juno's analog input signals, recommended signal processing circuitry, and extensive example schematics for various motion control applications.

14.1.1 CurrentA-D Signals

The *CurrentA-D* analog inputs are used with Juno's current control module. Depending on the motor type that is being driven (Brushless DC, DC Brush, or step motor) as few as two or up to all four of these signals are used.

The incoming voltage represents a bipolar leg current reading from 0.0V to 3.3V, with 0.0 representing the maximum negative reading, 1.65V representing a zero current reading, and 3.3V representing the maximum positive current reading. For detailed additional information on leg current sensing circuitry refer to the *MC78113 Electrical Specifications*.

Although Juno converts analog signals to digital via a 12 bit A/D, the numerical equivalent of the above input commands are scaled so that the maximum negative current feedback has a value of -20,480, a zero current has a value of 0, and the maximum positive current reading has a value of +20,480.

To read the raw analog value at the *CurrentA-D* pins the command **ReadAnalog** is used. The result is an unsigned number from 0 to 65,535. To read the value used for current loop processing the **GetFOCValue** command is used. The

difference between these two values is that the loop value has the calibration offset values applied and that the scaling changes by a factor of .625, from a range of -32,768 to +32,767 to -20,480 to +20,480. .

Example

Juno's current control module is used in an application to drive Brushless DC motors with up to 7.5 amps of peak current and 4 amps continuous. To provide the needed range margin (~30% above peak) the current sense resistors and associated analog processing circuitry are arranged to provide a full scale range of +/-10.0 amps, presenting a voltage of 0.0V for a reading of -10.0A, 1.65Vfor 0.0A, and 3.3V for +10.0A at the *CurrentA-D* pins.

With the current measurement circuitry scaled in this way the conversion from a measured current in amps to the raw numerical value read using the **GetFOCValue** command is value = I * 20,480 / 10.0A, and the conversion from measured current in amps to volts at the *CurrentA-D* pin is V = 1.65V + I * 1.65V / 10.0A.

To determine the numerical value for user specified current commands (provided via direct input SPI, or with commands such as **SetCurrent** and **SetMotorCommand**) we multiply by .625 / .50 reflecting the scaling within Juno's current loop. So given a desired current in amps the command numerical value is value = I * 1.25 * 32,767 / 10.0A, or value = I * 40,959 / 10.0A and conversely, given a commanded numerical value the equivalent commanded current in amps is I = value * 10.0A * .80 / 32,767, or I = value * 10.0A /40,959.

For information on calibrating the *CurrentA-D* and other analog inputs refer to Section 14.2, "Analog Signal Calibration." For scaling, filtering, and other important electrical details associated with the *CurrentA-D* inputs refer to the *MC78113 Electrical Specifications.*

Offset calibration of the analog CurrentA-D signals as well as (if used) the *AnalogCmd* and *Tachometer* signals is recommended for proper system performance.

14.1.2 AnalogCmd Signal

The *AnalogCmd* signal can be used to command an outer loop quantity, velocity, current (torque), or motor voltage. As for the *CurrentA-D* signals the voltage at the pin represents a bipolar command with 0.0V, 1.65V, and 3.3V representing the maximum negative, zero, and maximum positive commands respectively and with equivalent numerical values of -32,768, 0, and +32,767 respectively.

To read the raw analog value at the *AnalogCmd* signal the command **ReadAnalog** is used. With *AnalogCmd* selected as the loop command source, to read the value used as the loop command value the command **GetLoopValue** is used. The difference between these two values is that the loop value has the calibration offset value and Kanalogcmd gain applied (see next section for more information on Kanalogcmd).

14.1.2.1 AnalogCmd Gain

A special feature of the *AnalogCmd* signal is that its numerical value may be attenuated with a programmable gain factor. The register Kanalogcmd is used to optimize the commandable range when the *AnalogCmd* signal is used as the command source.

In an ideal system the full range of the *AnalogCmd* signal input, which is from 0.0V representing the most negative command value to 3.3V representing the most positive command value, should match the useable range of commandable values. The Kanalogcmd register, which is a 16 bit quantity with a range of 0 to 32,767 and a unity gain value of 32,767 (gain factor 1.0) facilitates that. The default value of Kanalogcmd is 16,384 (gain factor of .5).

The maximum recommended setting for Kanalogcmd is 1.0 except when *AnalogCmd* is used to command a velocity loop with an encoder or hall sensors providing the velocity feedback. In this case the maximum recommended setting

is 0.5. This is because velocity measurement by successive subtraction of positions can be relatively noisy, and therefore a lower gain is recommended to insure sufficient margin between the measured and commanded velocity.

To set Kanalogcmd the command **SetAnalogCalibration** is used, and to read this register back the **GetAnalogCalibration** command is used.

For examples of AnalogCmd scaling and usage see the next several sections.

14.1.2.2 AnalogCmd Signal With Position/Outer Loop

The *AnalogCmd* signal can also be used to command a pressure, flow rate, or other outer loop quantity value. In this mode the analog command interpretation is slightly different. Although voltage values of 0.0V, 1.65V, and 3.3V still generate numerical command values of -32,768, 0, and +32,767 respectively, the actual command outer loop quantity itself may or may not have a negative value depending on the nature of the outer loop measurement.

The *AnalogCmd* interpretation of voltages of 0.0V, 1.65V, and 3.3V therefore changes to: the smallest commandable value, the midpoint commandable value, and the largest commandable value. How the analog command value relates to the actual outer loop quantity is determined by the measurement feedback. The following example illustrates this.

Example

From the scaling established in the example in Section 14.1.3.2, "Tachometer Signal With Position/Outer Loop" and using a Kanalogcmd gain setting of unity (1.0), the *AnalogCmd* signal scaling will be the same as the measurement scaling with 0.0V commanding a pressure of 500 mbar, 1.65V commanding 1,000 mbar, and 3.3V commanding 1,500 mbar. Note that those commands are actually outside the application range of 650 mbar to 1,250 mbar and so with unity scaling of Kanalogcmd the smallest command presented at *AnalogCmd* should be 3.3V * (650 mbar - 500) / 1000 = .495 V, and the largest command should be 3.3V * (1,250 mbar - 500) / 1000 = 2.48 V.

Alternatively the 30% margin can be implemented by setting Kanalogcmd to a value of 0.7. This approach has the advantage that even if the *AnalogCmd* signal command voltage is at its maximum value the calculated value will be reduced by 30% thereby maintains the desired margin.

14.1.2.3 AnalogCmd With Velocity Loop

AnalogCmd can be used to command a velocity in connection with operation of the velocity loop module. Two examples are provided below, the first using Tachometer feedback to measure the velocity, and the second using an encoder.

Example 1

Following the velocity measurement scaling example provided in Section 14.1.3.1, "Tachometer Signal with Velocity Loop" the *AnalogCmd* signal will be used to command a desired velocity over the application range of -4,500 RPM to +4,500 RPM. To simplify the *AnalogCmd* circuitry we will take advantage of the Kanalogcmd gain value to implement the measurement to commandable safety margin. So we set Kanalogcmd to 32,768 * 4,500 / 6,500 = 32,768 * .692 = 22,686. With this gain setting of Kanalogcmd an applied voltage of 0.0V (largest negative value) commands a speed of -4,500 RPM, 1.65V commands a speed of 0 RPM, and +3.3V (largest positive value) commands a speed of +4,500 RPM.

Example 2

Rather than measure velocity with a tachometer, this same application detailed, in Section 14.1.3.1, "Tachometer Signal with Velocity Loop" will utilize an encoder with 2,048 counts per rotation to measure the velocity and the *AnalogCmd* signal to command the velocity. Because measurement of velocity by successive subtraction of positions may be noisy, two accomodations are made to manage this. The first is that the cycle time of the Juno's Velocity Loop is changed from its default value of 102 µsec (9,804 Hz) to a value of 1,024 µsec (977 Hz). The second is that the measurement range margin is increased to provide greater overhead. Rather than measuring a range of -6,500 RPM to +6,500 RPM thereby providing 30% overhead, the measured range will be -9,000 RPM to +9,000 RPM thereby providing 100% overhead from the desired commandable range of -4,500 RPM to +4,500 RPM.

With a cycle frequency of 977 Hz, and an encoder resolution of 2,048 counts/rot, 9,000 RPM corresponds to 9,000 rot/min * 2,048 cnts/rot / 60 sec/min = 307,200 cnts/sec. To convert to cnts per Juno cycle we use 307,200 cnts/sec / 977 cycles/sec = 314.43 cnts/cycle. The next step is to set a value for the velocity scalar. Although the Kanalogcmd gain gives us a lot of flexibility to set the velocity scalar we will use velocity scalar to scale the cnts/cyc up so that a maximum command at the SPI port (+32,767) correspond to 9,000 RPM. This is calculated with velocity scalar = 16,384/314.43 cnts/cycle = 52.107. Multiplying by 65,536 to reflect velocity scalar's unity value of 65,536 gives a velocity scalar setting of 52.107 * 65,536 = 3,414,884.

Once again, to simplify the *AnalogCmd* circuitry we will take advantage of the Kanalogcmd gain value to implement the measurement to commandable safety margin. So we set Kanalogcmd to 32,768 * 4,500 / 9,000 = 32,768 * 0.5 = 16,384. Since this is infact the default value for Kanalogcmd we don't have to change the value. In any case, with this gain setting an applied voltage of 0.0V (largest negative value) commands a speed of -4,500 RPM, 1.65V commands a speed of 0 RPM, and +3.3V (largest positive value) commands a speed of +4,500 RPM.

14.1.2.4 AnalogCmd With Current Loop

AnalogCmd can be used to command a current (torque) in connection with operation of the current loop module. The following example illustrates:

Example

Following the current loop scaling example provided in Section 14.1.1, "CurrentA-D Signals" a desired numerical current command value for a desired current in amps is calculated via: value = I * 1.25 * 32,767 / 10A. With a setting of the Kanalogcmd gain factor at unity (1.0) the *AnalogCmd* pin voltages will convert to numerical values over the range of -32,767 to +32,767 from lowest to highest command voltage. Therefore the expression for the current command in amps given an *AnalogCmd* signal voltage V is: I = (V - 1.65V) * 8.0A / 1.65V and conversely given a desired current in amps the command signal voltage will be V = 1.65V + I * 1.65V / 8.0A.

This means that with a unity setting of Kanalogcmd a voltage of 0.0V at *AnalogCmd* will command a current of -8.0A, a voltage of 1.65Vwill command 0.0A, and 3.3Vwill command +8.0A. Note that those commands are actually outside the application range of +/- 7.5A and so for this application the smallest *AnalogCmd* voltage command should be 1.65V + -7.5A * 1.65V /8.0A = .103V (which corresponds to a numerical value of -7.5A * 32,767 / 8.0A = -30,719) and the largest voltage command should be 1.65V + 7.5A * 1.65V / 8.0A= 3.197V (which corresponds to a numerical value of +7.5A *32,767 / 8.0A = +30,719).

Alternatively, and simplifying the implementation of the circuitry at the *AnalogCmd* pin, if a Kanalogcmd gain of .9375 is set (commanded value of 32,768 * .9375 = 30,720) this will allow the full range of 0.0V to 3.3V input settings to be input at *AnalogCmd* yet still generate the desired command range of -7.5A to +7.5A.

14.1.3 Tachometer Signal

The *Tachometer* signal can be used to provide motor velocity feedback for the outer loop or the velocity loop. Unlike the *CurrentA-D* and *AnalogCmd* signals the polarity of the *Tachometer* signal may be specified, meaning Juno can be programmed to handle feedback signals that increase in voltage with increases in value as well as decrease in voltage with increases in value.

To select the *Tachometer* signal along with its polarity as the feedback source the **SetLoop** command is used. To read the selected value back the **GetLoop** command is used. For a positive polarity setting incoming voltages of 0.0V, 1.65V, and 3.3V represent the maximum negative, zero, and maximum positive velocity readings respectively. For a negative polarity setting incoming voltages of 0.0V, 1.65V, and 3.3V represent the maximum positive, zero, and maximum negative velocity readings respectively.

To read the raw analog value at the *Tachometer* signal the command **ReadAnalog** is used. The values read back are not altered by the polarity setting described above. With *Tachometer* selected as the loop feedback source to read the loop feedback value (which includes the polarity adjustment as well as offset calibration) the command **GetLoopValue** is

used. The difference between these two values is that the loop value has the offset value, the polarity change if applicable, and the effect of the biquad filter applied.

14.1.3.1 Tachometer Signal with Velocity Loop

The following example provides an illustration of signal scaling when the *Tachometer* signal is used for velocity measurement.

Example

Juno's velocity control module is used in an application with a tachometer to drive a DC Brush motor at up to 4,500 RPM (revolutions per minute). To provide the needed range margin (~30% above peak expected speed) the tachometer and associated analog processing circuitry are arranged to provide a full scale range of +/-6,500 RPM, presenting a voltage of 0.0V for a reading of -6,500 RPM, 1.65Vfor 0 RPM, and 3.3Vfor +6,500 RPM at the ***Tachometer*** pin.

With the velocity measurement circuitry scaled in this way the conversion of the numerical command to velocity (V) in RPM is V = value * 6,500 RPM / 32,767, the conversion from a desired velocity in RPM to the numerical value is value = V * 32,767 / 6,500 RPM, and the conversion from velocity (Vel) to volts at the ***Tachometer*** pin is V = 1.65V + Vel * 1.65V / 6,500 RPM.

14.1.3.2 Tachometer Signal With Position/Outer Loop

As was the case for the ***AnalogCmd*** signal, the ***Tachometer*** signal, when used to provide the outer loop measurement, should be interpreted such that 0.0V, 1.65V, and 3.3Vrepresent the smallest measurable value, the midpoint of measurable values, and the largest measurable value.

Example

Juno's outer loop controller is used in an application to precisely control air pressure within a process chamber from 650 to 1,250 mbar. To provide the needed range margin (30% or so) the feedback pressure sensor is set up to have a scale of 500 mbar to 1,500 mbar, presenting a voltage of 0.0V for a reading of 500 mbar, 1.65Vfor 1,000 mbar, and 3.3Vfor 1,500 mbar. With the pressure reading analog circuitry scaled in this way the conversion of the numerical command to pressure is therefore P in mbar = 1,000 + 500 * value / (32,767), and the conversion of pressure to volts is V = 3.3V * (P-500) / 1,000.

14.1.4 Temperature, BusCurrentSupply, and BusVoltage Signals

These analog input signals are used with Juno's DC Bus/amplifier related safety check features. Similar to other analog inputs they expect input voltages in the range of 0.0V to 3.3V. Unlike these other signals however these signals are unipolar, encoding only the magnitude of the measured quantity.

Although the mechanism for selecting the polarity interpretation is different than for the ***Tachometer*** signal, the ***Temperature*** signal provides a settable polarity so that both temperature increasing/voltage increasing and temperature increasing/voltage decreasing sensor types can be accommodated. The ***BusCurrentSupply*** and ***BusVoltage*** signals always expect positive encoding, such that higher voltages encode higher current or DC bus voltage readings.

For more information on the functioning of these signals refer to Chapter 11, *Amplifier Safety & DC Bus Monitoring*, as well as to the *MC78113 Electrical Specifications*.

14.2 Analog Signal Calibration

After integration into a particular PCB (printed circuit board), for best performance, it is generally recommended that if used, the *CurrentA-D*, *AnalogCmd*, and *Tachometer* signal analog input offsets be calibrated so that their zero value readings are as close to 1.65V as possible.

Whether or not calibration is needed in a particular application depends on the external analog signal processing circuitry used and the extent to which the absolute best motion performance, particularly motor smoothness and quietness, is important. Higher precision external circuitry or use of external offsetting circuitry may obviate the need for internal calibration. Conversely Juno's internal analog offset calibration procedure may allow less expensive circuitry to be used.

For the *Temperature* and *BusVoltage* signals, offsetting is allowed but generally not necessary. The *BusCurrentSupply* signal does not support an offset calibration nor is one needed for proper functioning.

Calibration of Juno analog inputs should occur when the board is powered up, with no analog velocity or torque command asserted, with no motor output command asserted, and with the physical motor axis stationary. Two overall calibration methods are provided. The simplest method is to send a **CalibrateAnalog** command. This command allows one or more signals to be calibrated. This command will automatically measure and set the offsets so that the leg current analog, tachometer and AnalogCmd inputs are zeroed out. Because a number of samples are taken and averaged, 100 mSec should be allowed for this command to complete. In addition, the Juno operating mode should be set to motor output only before this command is executed.

The second method is to directly read each analog input via the **ReadAnalog** command and then write the same values for the corresponding analog offsets using the **SetAnalogCalibration** command. When using this manual method it is recommended that a number of analog reads of each signal are averaged together to improve the offset accuracy.

Regardless of how the analog offsets are determined, unless explicitly stored into NVRAM they will not be retained after a reset or power cycle. For more information on NVRAM configuration storage see Chapter 12, *Power-up, Configuration Storage & NVRAM.*

Calibration of Juno's CurrentA-D, AnalogCmd, and Tachometer inputs, if used, is recommended to achieve the full extent of smooth and quiet motion that Juno is capable of.

14.2.1 Analog Signal Calibration in the Production Application

After integration into a PCB (printed circuit board), it is recommended that the Juno analog inputs be calibrated. While some applications will not need to worry about these calibrations, for performance intensive applications where the quietest, smoothest, and most accurate motion is desired, calibration of the analog inputs is recommended. For more information refer to Section 2.9.2, "Analog Signal Calibration in the Production Application."

14.3 Direct Input SPI (Serial Peripheral Interface)

Juno provides an SPI port that may either be used for high level host communications, for direct input of commands, or for direct input of feedback values in conjunction with control loop operation. For more information on using the

SPI port for host communications, including how to command that SPI port to function as a direct SPI port, refer to Section 13.5, "SPI (Serial Peripheral Interface) Communications."

There are two ways to set Juno's SPI port for direct input SPI operation. The first is via the **SetCommMode** host command sent while the SPI port is operated in host communications mode. Immediately upon sending this command the SPI port will thereafter switch to direct input mode, thus rendering SPI host communications inoperable, however see Section 14.3.3, "Resetting SPI Direct Input Mode" for information on restoring the SPI port to host communications while in direct input mode. The second occurs automatically if the loop command source is set to the SPI port, or if the outer loop feedback source is set to SPI port. For more information on setting the loop command source refer to Section 3.1, "Position Loop Operation", Section 4.1, "Selecting the Command Source" or Section 5.1, "Selecting the Command Source." For information on setting the outer loop feedback source to SPI refer to Section 3.2, "Outer Loop Operation."

Once the SPI port enters direct input mode it will remain there until the next power cycle, or until a special SPI port reset sequence is executed. See Section 14.3.3, "Resetting SPI Direct Input Mode" for more information. If an NVRAM initialization sequence has been entered which sets the SPI port to direct input mode than only a reprogramming of that NVRAM sequence, or application of the SPI Port reset sequence can restore the SPI port to host communication mode.

14.3.1 Direct Input SPI Data Write

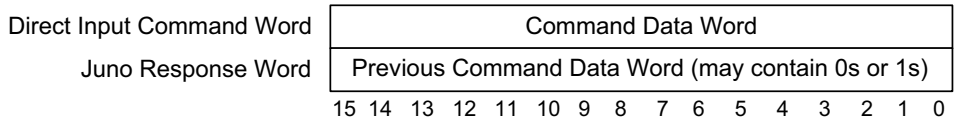


Figure 14-1: Direct Input SPI Format

The signals that are used in conjunction with the SPI port are *SPIXmt*, *SPIRcv*, *SPIClock*.

The only SPI port operation supported in connection with the direct input function is a write by the external circuitry of a 16 bit command data word. The write format is shown in Figure 14-1. In the direct input SPI mode the external circuitry serves as the SPI master generating the clock and the enable, and transmitting the 16-bit data word data to Juno.

The returned word by Juno is the previous command word received. It is recommended, but not required, that the external circuitry read this returned word and confirm that it matches the previously transmitted word.

14.3.2 SPI Direct Input Data Formats

The interpretation of the transmitted SPI command data word that is received and interpreted by Juno depends on the control mode that Juno has been set to. The following table shows this, indicating the content and interpretation of the transmitted data word for each use of the direct input SPI port.

Function	Control Module	Interpretation
Velocity command	Velocity loop	Signed 16 bit value representing the velocity loop command value
Velocity command	Position/outer loop	Signed 16 bit value representing a velocity command when using the position loop.
Current command	Current loop	Signed 16 bit value representing the current loop command value
Voltage command	Motor output	Signed 16 bit value representing the motor out module command value
Position increment command	Step motor control	Signed 16 bit value representing the relative (incremental) distance that the position command has changed since the previous relative position command write.

Outer loop command	Position/outer loop	Signed 16 bit value representing the outer loop command, typically a pressure or temperature.
Outer loop measurement	Position/outer loop	Signed 16 bit value representing the measured outer loop value, typically a pressure or temperature reading.

Direct input SPI data for the velocity, current, or step motor control modules is signed and generally represents a bipolar measured quantity centered around a value of zero.

Direct input SPI data used for the position/outer loop module is signed, however depending on how the feedback circuitry has been scaled the measurement sensor value may or may not be bipolar. See section, Section 14.1.2.2, "AnalogCmd Signal With Position/Outer Loop" for more information and a detailed example.

14.3.3 Resetting SPI Direct Input Mode

For Juno ICs that contain an initialization sequence in the NVRAM that sets the SPI host port to direct input mode there may be situations where it is desirable to reset the SPI port to host communications mode, thereby allowing the NVRAM programming to be changed or allowing other IC changes to be made via host communications.

Exiting the host SPI port from direct input mode and restoring to host communication mode can be accomplished by sending a special three word write sequence; A 0x55AA followed by a 0x33CC followed by a 0x0FF0. After receiving this sequence of direct input writes via the SPI port Juno will generate a drive exception error in the Event Status register and a bit in the Drive Fault Status register will indicate an SPI mode change. In addition the SPI mode will be set to host communications.

This SPI mode reset sequence does not affect the NVRAM content, so if the NVRAM initialization sequence sets the SPI mode to direct input, at the next power cycle the SPI port will revert to direct input mode.

Index

N

O

P

R

S

T

V

W