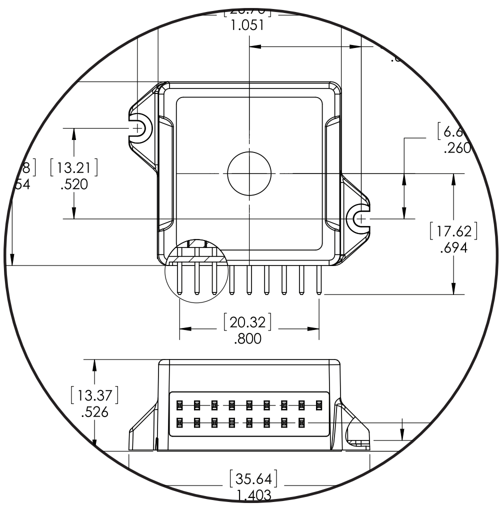



# Atlas Digital Amplifier

## Complete Technical Reference

Revision 2.2/ June 2024

**Performance Motion Devices, Inc.**

80 Central Street, Boxborough, MA 01719

www.pmdcorp.com

## 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

**Atlas Digital Amplifier User Manual**

Description of the Atlas Digital Amplifier electrical and mechanical specifications along with a summary of its operational features.

**Magellan Motion Control IC User Guide**

Complete description of the Magellan Motion Control IC features and functions with detailed theory of operation.

**DK58113 Developer Kit User Manual**

How to install, configure, and operate the DK58113 Family IC developer kits.

**DK58420 Developer Kit User Manual**

How to install, configure, and operate the DK58420 and DK55420 developer kits.

# Table of Contents

# List of Figures

# 1. Introduction

1

## *In This Chapter*

- Atlas Digital Amplifier Overview
- Typical Applications
- Features and Functions
- Atlas Developer Kits

## 1.1 Atlas Digital Amplifier Overview

Atlas Digital Amplifiers are single-axis amplifiers that provide high performance torque control of Brushless DC, step motor, and DC Brush motors. They accept digital torque commands from an external source and are used directly for motor torque control applications, or in conjunction with higher level controllers for velocity or positioning applications. Their very compact size and range of power output levels make them an ideal solution for single-card machine controllers that require high performance in a small envelope.

Atlas digital amplifiers provide many advanced control features including user-programmable gain parameters, performance trace, field oriented control, and $I^2t$ current management. Atlas amplifiers are powered from a single supply voltage, and provide automatic protection from overcurrent, undervoltage, overvoltage, overtemperature, and short circuit faults.

The Atlas digital amplifier family has been designed to work seamlessly with PMD's Magellan family of motion control ICs. Alternatively, they can be used with dedicated FPGAs, digital signal processors, or general purpose microprocessors. Communication to/from Atlas amplifiers is via SPI (Serial Peripheral Interface) using a simple, packet-oriented protocol. For step motors, in addition to the SPI format a dedicated pulse & direction input mode is provided.

Atlas amplifiers are packaged in plastic and metal solderable modules and are available in an ultra compact package size with a total footprint of 1.4 $inch^2$ (9.0 $cm^2$) and a compact package size with a footprint of 2.6 $inch^2$ (16.8 $cm^2$). They come in three power levels; 75 watts, 250 watts, and 500+ watts and utilize standard through-hole pins for all electrical connections.

Atlas amplifiers are provided in both vertical and horizontal mounting configurations and have integral attachment tabs to allow for a variety of mechanical mounting and heat sink options. The following table shows the available configurations of the Atlas Digital Amplifiers:

| P/N | Power Level (continuous) | Voltage | Size | Mounting Style | Motor Type |
|---|---|---|---|---|---|
| **Step Motor** | | | | | |
| MD241048/02VB | Low (75W) | 12-48V | Ultra compact | Vertical | Step Motor |
| MD241048/02HB | Low (75W) | 12-48V | Ultra compact | Horizontal | Step Motor |
| MD241048/05VB | Medium (250W) | 12-48V | Ultra compact | Vertical | Step Motor |
| MD241048/05HB | Medium (250W) | 12-48V | Ultra compact | Horizontal | Step Motor |
| MD141056/25VB | High (500+W) | 12-56V | Compact | Vertical | Step Motor |
| MD141056/25HB | High (500+W) | 12-56V | Compact | Horizontal | Step Motor |

| P/N | Power Level (continuous) | Voltage | Size | Mounting Style | Motor Type |
|---|---|---|---|---|---|
| **Brushless DC** | | | | | |
| MD231048/02VB | Low (75W) | 12-48V | Ultra compact | Vertical | Brushless DC |
| MD231048/02HB | Low (75W) | 12-48V | Ultra compact | Horizontal | Brushless DC |
| MD231048/05VB | Medium (250W) | 12-48V | Ultra compact | Vertical | Brushless DC |
| MD231048/05HB | Medium (250W) | 12-48V | Ultra compact | Horizontal | Brushless DC |
| MD131056/25VB | High (500+W) | 12-56V | Compact | Vertical | Brushless DC |
| MD131056/25HB | High (500+W) | 12-56V | Compact | Horizontal | Brushless DC |
| **DC Brush** | | | | | |
| MD211048/02VB | Low (75W) | 12-48V | Ultra compact | Vertical | DC Brush |
| MD211048/02HB | Low (75W) | 12-48V | Ultra compact | Horizontal | DC Brush |
| MD211048/05VB | Medium (250W) | 12-48V | Ultra compact | Vertical | DC Brush |
| MD211048/05HB | Medium (250W) | 12-48V | Ultra compact | Horizontal | DC Brush |
| MD111056/25VB | High (500+W) | 12-56V | Compact | Vertical | DC Brush |
| MD111056/25HB | High (500+W) | 12-56V | Compact | Horizontal | DC Brush |
| **Multi-Motor** | | | | | |
| MD281048/02VB | Low (75W) | 12-48V | Ultra compact | Vertical | Multi-motor* |
| MD281048/02HB | Low (75W) | 12-48V | Ultra compact | Horizontal | Multi-motor* |
| MD281048/05VB | Medium (250W) | 12-48V | Ultra compact | Vertical | Multi-motor* |
| MD281048/05HB | Medium (250W) | 12-48V | Ultra compact | Horizontal | Multi-motor* |
| MD181056/25VB | High (500+W) | 12-56V | Compact | Vertical | Multi-motor* |
| MD181056/25HB | High (500+W) | 12-56V | Compact | Horizontal | Multi-motor* |

**Multi-motor motor type allows the Atlas to be configured by the user to drive either Step Motor, Brushless DC, or DC Brush motor type.*

This manual provides a complete description of the electrical and mechanical specifications for the Atlas Digital Amplifiers, along with a detailed description of its operational features. For more information on the Magellan Motion Control IC consult the *Magellan Motion Control IC User Guide*.

# 1.2 Typical Applications

The following section provides overview diagrams of typical applications utilizing the Atlas amplifier products.

## 1.2.1 Single Axis Positioning Motion Controller

The diagram below shows a PMD MC58113 Motion Control IC sending a stream of commands to an Atlas Amplifier to provide positioning control of a Brushless DC, DC Brush, or Step Motor.

For Brushless DC motors the data being sent consists of the commanded torque and the commutation angle. For step motors the transmitted data consists of the microstepping angle. For DC Brush motors the data consists of the commanded torque.

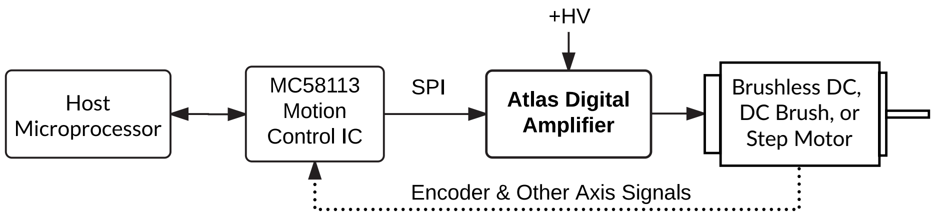



**Figure 1-1: Single Axis Magellan With Atlas Amplifier**

## 1.2.2 Multi Axis Positioning Motion Controller

The diagram below shows a PMD Magellan MC58000 series or MC55000 series multi-axis motion control IC being used with two or more Atlas Amplifiers to provide control of Brushless DC, DC Brush, or Step Motors in a positioning application. If desired each axis can control a different motor type, so that, for example, Brushless DC motors can be used along with step motors in the same controller.

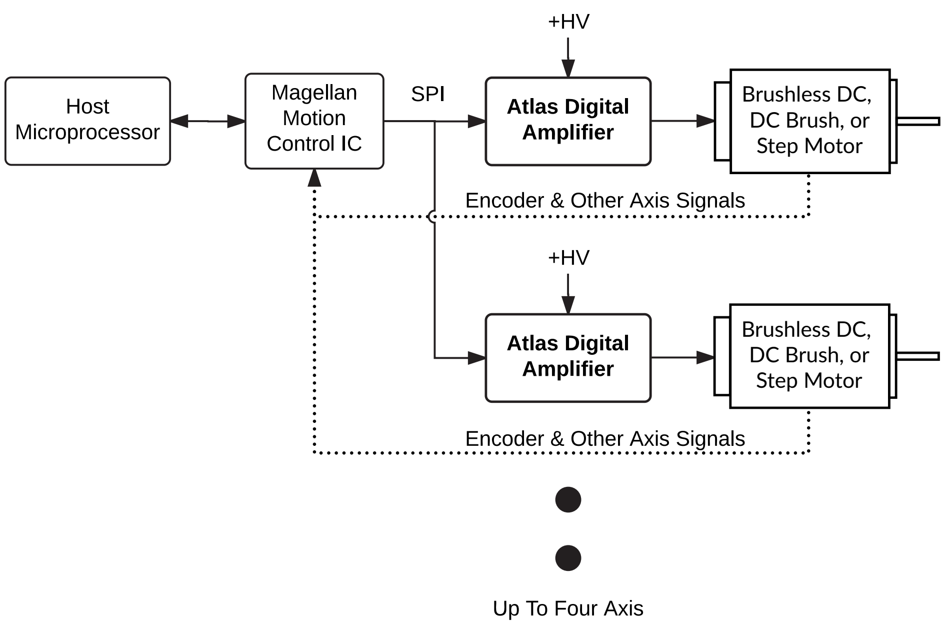



**Figure 1-2: Multi Axis Magellan With Atlas Amplifiers**

## 1.2.3 Microprocessor-Based Motion Controller

The diagram below shows the Atlas Amplifier being driven by a general purpose microprocessor that provides high level path generation and servo loop closure and outputs continuous desired torque commands or desired position increments for step motors to the Atlas Amplifier via the SPI (Serial Peripheral Interface).

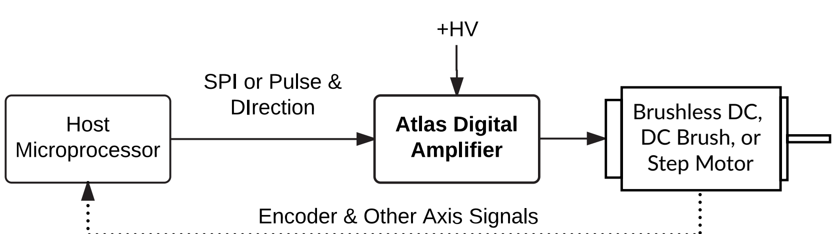
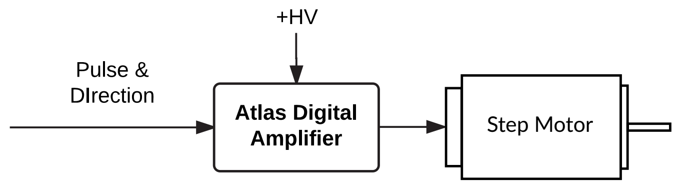



**Figure 1-3: Direct Host Microprocessor With Atlas Amplifiers**

### 1.2.4 Stand Alone Step Motor Amplifier

The diagram below shows the Atlas Amplifier being directly driven by pulse & direction signals. These signals may come from a microprocessor, a control card, or any other existing motion control device that outputs pulse & direction signals. In this mode the Atlas unit operates 'stand-alone,' and utilizes configuration control parameters previously stored into the Atlas unit's NVRAM (non-volatile) memory.

**Figure 1-4: Direct Host Microprocessor With Atlas Amplifiers**




There are a few options for configuring Atlas units for stand alone operation:

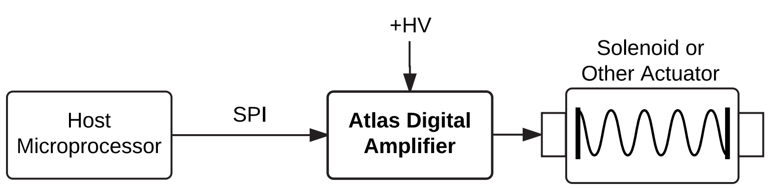
- Pro-Motion can be used with the Atlas Developer Kit to program Atlas units
- The user can develop their own NVRAM programming system by utilizing the SPI (Serial Peripheral Interface) Atlas command protocol. For more information refer to the *Atlas Digital Amplifier Complete Technical Reference.*

### 1.2.5 Force Control

The Brushless DC and DC Servo Atlas units can be used for general purpose force control applications such as remote teleoperation, force feedback, solonoid actuation, and any other general purpose valve/actuator control where a precisely controllable current is needed.

In this application the torque command may be sent continuously by the host microprocessor or from time to time as required by the application. In either case the Atlas provides very accurate current/torque control resulting in smooth and precise application of force.

**Figure 1-5: Atlas Force Control**




## 1.3 Features and Functions

The Atlas family of amplifiers provide an extensive list of functions, including:

- Available in Brushless DC, DC Brush, Step Motor, and multi-motor motor types
- High performance all-digital power amplifier
- Works with Magellan ICs, FPGAs or microprocessor-based controllers
- Digital SPI interface eliminates analog +/- 10V signals
- Available in 75W, 250W, and 500W+ power levels
- Rugged plastic solderable module format uses standard through-hole pins

- Available in ultra compact 1.05" x 1.05" x .53" (27mm x 27mm x 13mm) size or compact 1.52" x 1.52" x .60" (39mm x 39mm x 15mm) size
- Horizontal and vertical mount configurations
- Includes rugged mechanical tab mounts
- Supply voltage range of 12V up to 56V
- High current output up to 14A continuous, 25A peak
- Digital current loop with choice of standard A/B or Field Oriented Control (FOC)
- Direct signal pulse and direction input
- $I^2t$ current foldback limiting
- Overcurrent, overvoltage, undervoltage, overtemperature, and SPI command watchdog timeout protection
- Single DC supply operation.
- Enable input and FaultOut output safety interlock signals
- SPI (Serial Peripheral Interface) up to 8 MHz
- Performance trace of up to 1,020 words and four simultaneous variables
- 1,024 word non-volatile parameter storage
- Microstepping control with up to 256 microsteps per full step
- Signal conditioning buffers and analog filters on all I/O signals
- Fully RoHS compliant and CE marked

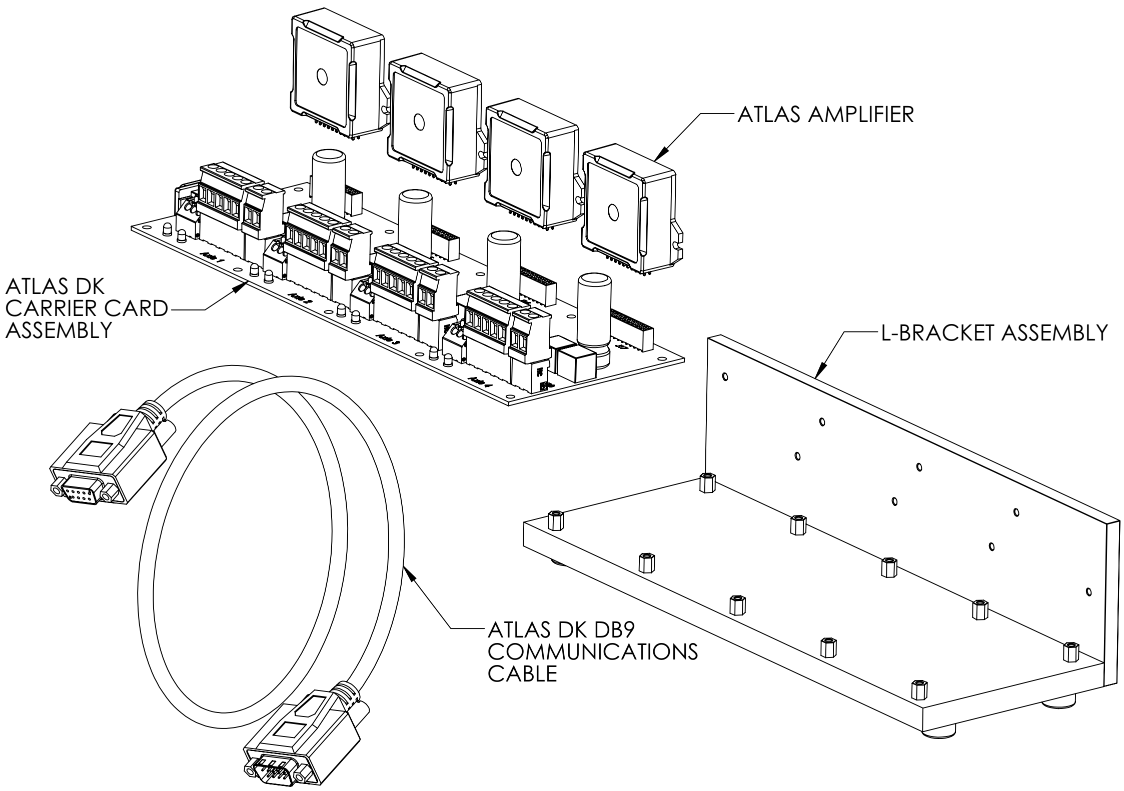
# 1.4 Atlas Developer Kits

To simplify development, four different Atlas Developer Kits are available, reflecting a choice of one or four axis board, and a choice of two different Atlas mounting configurations; vertical and horizontal. The following table shows this:

| Developer Kit P/N | # of Axes | Atlas Type |
|---|---|---|
| MDK1LI0000V | 1 | Vertical |
| MDK1LI0000H | 1 | Horizontal |
| MDK4LI0000V | 4 | Vertical |
| MDK4LI0000H | 4 | Horizontal |

Figure 1-6 shows an overview of an Atlas Developer Kit assembly. The particular assembly shown is for a four axis vertical DK, but the overall elements are similar for one axis developer kits. Horizontal developer kits are also similar except that there is no vertical plate included. Note that the Atlas units shown in the figure are not included with the developer kit and must be purchased separately.

**Figure 1-6: Developer Kit Components**




The following software and hardware components are included in every Atlas Developer Kit:

- Pro-Motion Windows-based exerciser software
- C-Motion SDK
- PDFs of all documentation
- Atlas DK DB9 communications cable

The Atlas DK boards are designed for direct use with the compact Atlas format however each DK includes converter cards that allow the ultra compact Atlas to be plugged into the compact Atlas DK board socket.

Vertical Atlas DKs include an L-bracket vertical plate which provides a stable mechanical base from which you can operate your prototype system motors. With the vertical plate, the Atlas units have additional heat sinking, which can be extended further by connecting the vertical plate to your own heat sink or cold plate. Horizontal Atlas DKs utilize individual heat sinks which are included with the DK.

Electrical connection to the Atlas DK board is made by DB9 connector, and by terminal screw connectors. Designers who plan to use the Atlas in conjunction with PMD's Magellan Motion Control ICs can connect the Atlas DK to the MC58113 or Magellan DK card, purchased separately. For more information on these products see the *DK58113 Developer Kit User Manual* or the *DK58420 Developer Kit User Manual*.

Refer to Appendix A, *Atlas Developer Kits* for complete information on setting up and operating the Atlas DKs.

# 2. Functional Characteristics

2

## *In This Chapter*

- Operational Specifications
- Physical Dimensions
- Mechanical Mounting Options

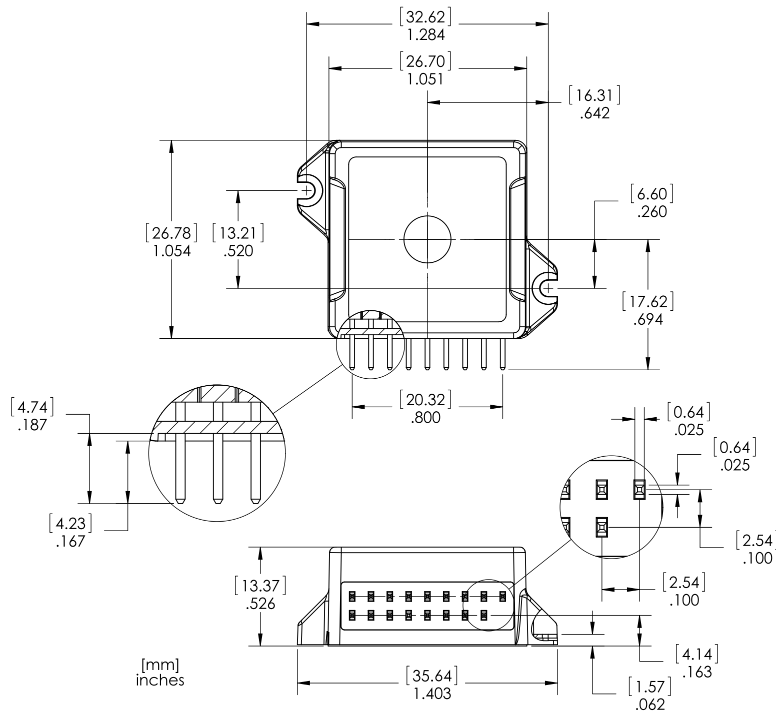
## 2.1 Operational Specifications

| Operating Parameter | Value |
|---|---|
| Motor types supported: | Brushless DC, DC Servo, Step Motor |
| Communication format: | SPI (Serial Peripheral Interface) |
| SPI clock frequency range: | 2.0 MHz to 8.0 MHz |
| Torque command rate: | up to 9.7 kHz |
| Current measurement resolution: | 12 bits |
| Current loop type: | P, I (proportional, integral) with Integral limit |
| Current loop resolution: | 16 bits |
| Current loop rate: | 19.530 kHz |
| Current loop modes: | A/B, field oriented control, third leg floating |
| Safety functions: | over current detect, programmable over temperature detect, programmable overvoltage detect, programmable under voltage detect, programmable $I^2t$ current foldback, SPI command watchdog timeout |
| Output limiting: | Programmable $I^2t$ energy, current, and voltage limit |
| Command modes: | SPI voltage, SPI torque, pulse & direction signal |
| PWM rate: | 20 kHz, 40 kHz, 80 kHz, or 120 kHz |
| PWM generation modes: | sinusoidal, space vector modulation, standard single-phase |
| Pulse & direction rate: | 1.0 M Pulses/sec |
| Microsteps per full step: | up 256 per full step |
| Trace capture modes: | one time, rolling-buffer |
| Trace trigger modes: | internal clock, external by controller |
| Trace buffer size: | 1,020 16-bit words |
| NVRAM storage size: | 1,024 16-bit words |

# 2.2 Physical Dimensions

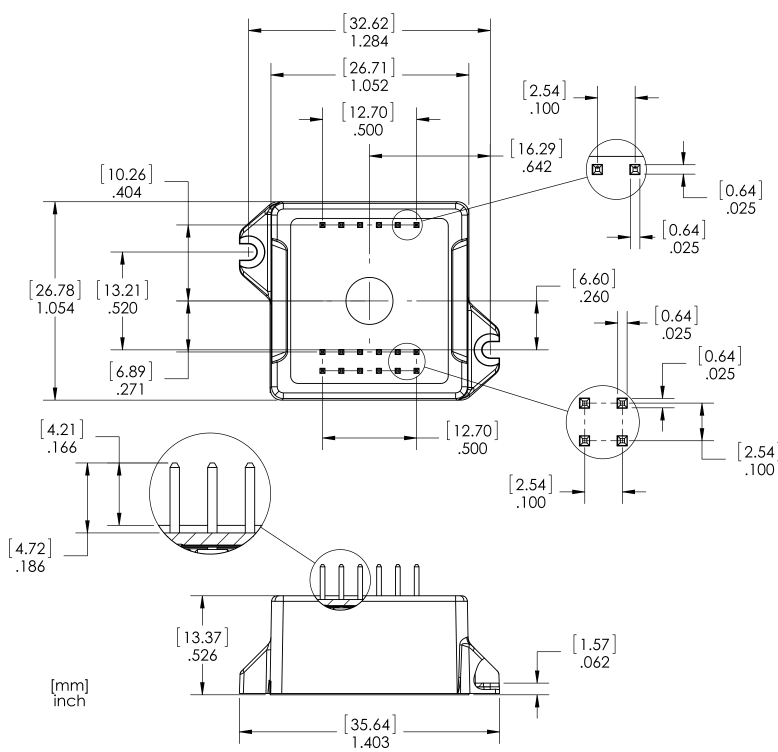
## 2.2.1 Vertical Unit, Ultra Compact Package

**Figure 2-1: Vertical Unit - Ultra Compact Package**




## 2.2.2 Horizontal Unit, Ultra Compact Package

**Figure 2-2: Horizontal Unit - Ultra Compact Package**



## 2.2.3 Vertical Unit, Compact Package

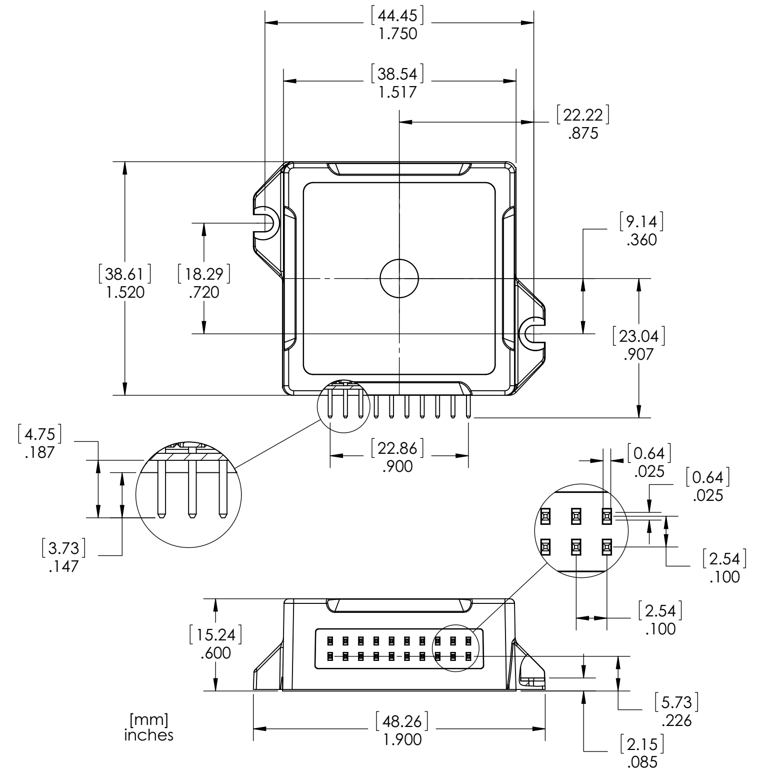



**Figure 2-3: Vertical Unit - Compact Package**

## 2.2.4 Horizontal Unit, Compact Package

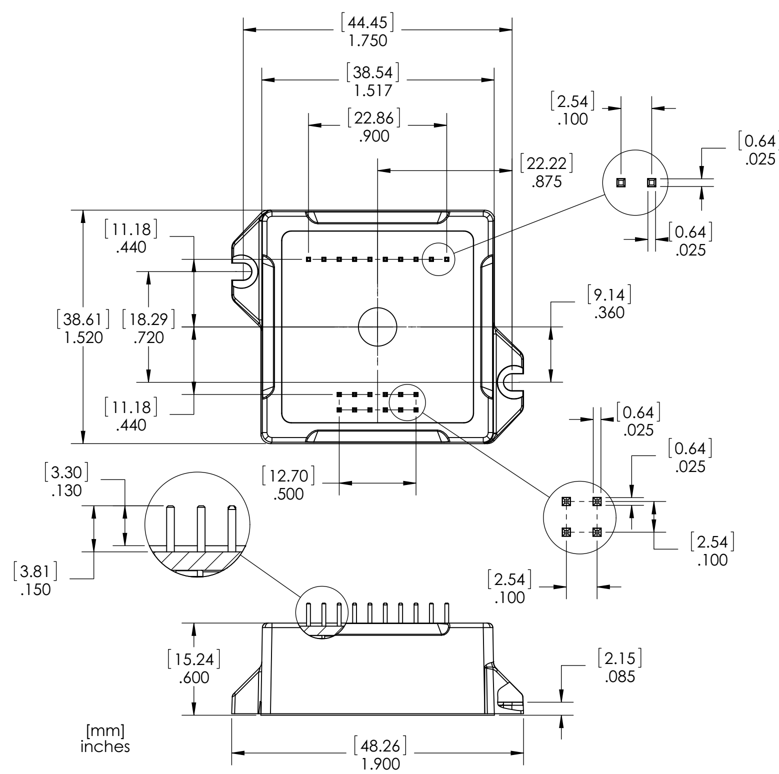



**Figure 2-4: Horizontal Unit - Compact Package**

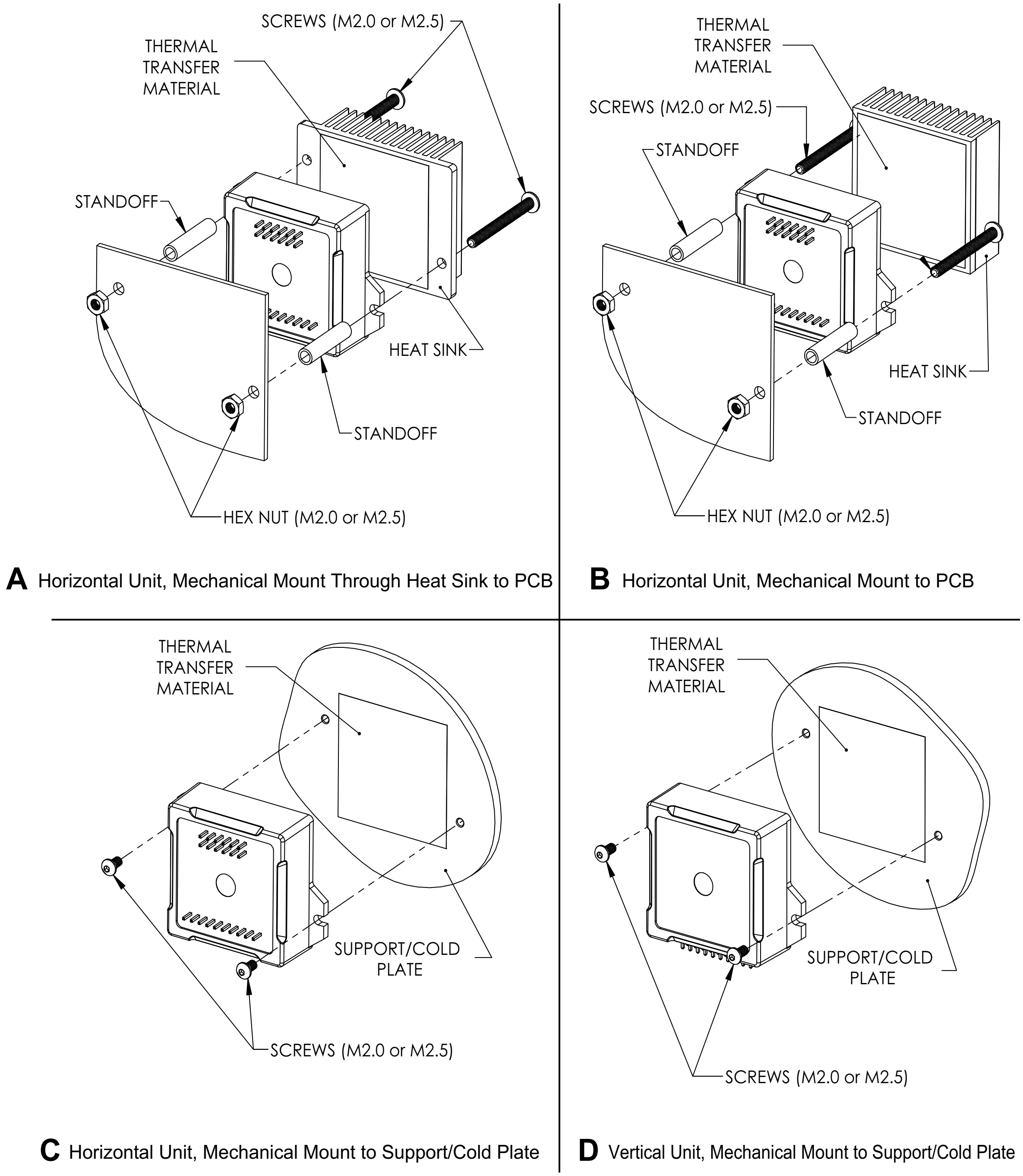
# 2.3 Mechanical Mounting Options

Atlas amplifiers are provided in two separate package sizes, ultra compact and compact, and in two separate mounting configurations; vertical and horizontal. There are some very low power applications where the Atlas unit may be mounted without mechanical attachment to the screw tabs. In such cases mechanical attachment to the PCB occurs via the electrical solder connections.

Most applications however will utilize the Atlas unit's integral screw tab mounts to rigidly connect the Atlas to the PCB, to a heat sink, or to some other mechanical support. As shown in Figure 2-5 there are a number of Atlas mounting options available when using the Atlas screw tabs. The choice of the mounting hardware depends on the demands of the application.

The following table provides information related to the mechanical screw tab mounts:

| Atlas Package | Recommended screw type | Maximum screw head diameter | Maximum screw body diameter |
|---|---|---|---|
| Ultra Compact | M2.0 | 4.2 mm | 2.2 mm |
| Compact | M2.5 | 5.4 mm | 2.8 mm |




**A** Horizontal Unit, Mechanical Mount Through Heat Sink to PCB

**B** Horizontal Unit, Mechanical Mount to PCB

**C** Horizontal Unit, Mechanical Mount to Support/Cold Plate

**D** Vertical Unit, Mechanical Mount to Support/Cold Plate

**Figure 2-5: Horizontal & Vertical Unit Mounting Options**

## 2.3.1 Mounting Guidelines

Atlas amplifiers, while designed to be robust and easy to install, contain active electronics that can only function reliably when the mechanical integrity and operating environment of the Atlas is maintained. The next three sections provide important recommendations and guidelines for the configuration, selection, placement, mounting method, and installation procedure for Atlas amplifiers.

*Choice of vertical or horizontal Atlas*. The horizontal configuration of Atlas is recommended for applications where the Atlas is not mechanically mated to a supporting plate and where vibration or movement-related forces may be present. When the Atlas unit is mechanically mated to a supporting plate, either the horizontal or the vertical configuration may be used. Figure 2-5C and Figure 2-5D show the Atlas unit mechanically mated to a supporting plate.

*Attaching Atlas to a supporting plate*. Some Atlas applications will utilize a supporting plate for heat removal or for enhanced mechanical stability. For Atlas installations that may be subject to vibration or movement-related forces and that utilize a supporting plate, special care should be taken to insure that there is no movement between the circuit card that the Atlas is soldered or socketed to and the supporting plate which the Atlas is mechanically attached to. Such movement could result in damage to the Atlas unit, the circuit card, or the supporting plate.

*Attaching Atlas to a free-standing heatsink*. Some Atlas applications will utilize a free standing heat sink, such as is shown in Figure 2-5A and Figure 2-5B. Free standing heat sinks are recommended with horizontal Atlas units but are not recommended for use with vertical Atlas units. When mounting Atlas units with free standing heat sinks special care should be taken where vibration or movement-related forces may be present. These forces, acting on the additional mass of the heat sink, may impart excessive mechanical stress on the Atlas resulting in damage to the Atlas unit, the circuit card, or the heat sink. Depending on the nature and magnitude of the forces, in these applications mounting the Atlas to a supporting plate may be preferred.

*Choice of socket or solder connection to the circuit card*. For best electrical contact to the printed circuit board (PCB), connection by soldering to the Atlas is generally recommended. This is particularly true for Atlas units that are not mated to a supporting plate. When the Atlas unit is mounted to a supporting plate either solder or socket electrical connections may be used, with solder connections recommended for applications benefitting from rigid connection of the Atlas to the PCB, and sockets being recommended when greater mechanical isolation of the PCB from the mechanical support is desired.

> Some of the electrical ratings of the Atlas may not be achievable when electrical connection to the Atlas is via sockets rather than via soldering. It is the responsibility of the user to determine whether a particular motor output current and voltage rating may be achieved with a given socket.

## 2.3.2 Thermal Transfer Materials

Thermal transfer materials in the form of thermal tape, pads, paste, or epoxy may be used to improve thermal transfer between the Atlas' metal plate and an attached heat sink or supporting plate. These materials improve thermal conductivity by filling in air gaps that form when two metallic surfaces are mated.

Figure 2-5 shows a typical application of a thermal transfer material between the Atlas and a heat-removing metal surface. The following guidelines may be helpful in selecting and sizing the thermal transfer material best-suited to your application.

The capacity of thermal transfer materials to transfer heat (known as the bulk conductivity) is much lower than that of metals such as aluminum or copper. Therefore, in general, the thinner the transfer material the better. Thickness of the material is only precisely controllable for thermal pads and thermal tapes, with thermal pads providing the thinnest available interfaces beginning at 5 mils (.127 mm) or even less. For use with Atlas amplifiers thermal transfer materials that are thicker than 40 mils (1.0 mm) are not recommended regardless of the material used.

When using thermal paste or thermal epoxy glue the thickness should be carefully controlled via a silk screen or other wet film application process. The Atlas unit itself should not be used to squeeze non-uniformly applied paste or epoxy flat during installation. Doing so may result in damage to the Atlas.

Whether using tape, pads, paste, or epoxy, as shown in Figure 2-6, the thermal transfer material that is used as the interface should not extend to the area under the Atlas' tabs because this may reduce the amount of compression that occurs in the thermal transfer area. The following table provides dimensions for the applied thermal transfer material for the two available Atlas package sizes:

| Atlas Package Size | Maximum Pad Dimensions |
|---|---|
| Ultra Compact | .1.00" x .95" (25.4 mm x 24.1mm) |
| Compact | 1.40" x 1.50" (35.6 mm x 38.1 mm) |

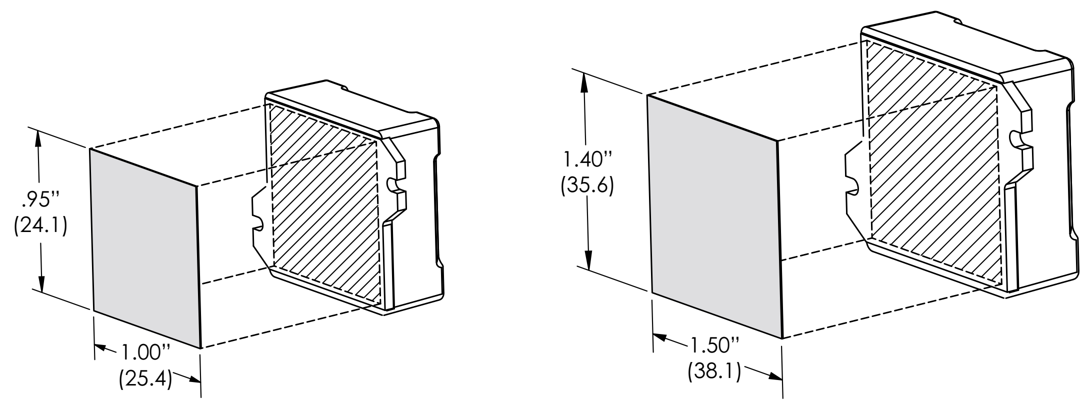



**Figure 2-6: Recommended Atlas Unit Thermal Transfer Material Dimensions**

## 2.3.3 Atlas Installation

There are a number of precautions and procedures that should be followed to maintain the electrical and mechanical integrity of the Atlas unit during installation.

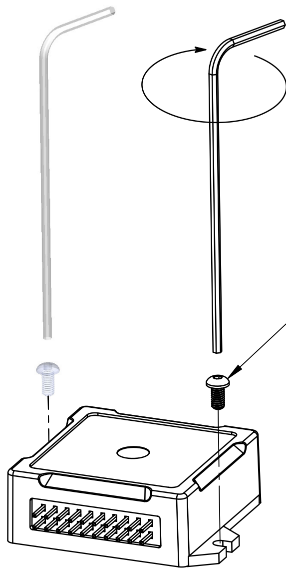
*Soldering Atlas units in place.* Applications that utilize Atlas units that are not mechanically mated to a heat sink or that are mated to a self-standing heat sink may utilize a standard soldering process without special precautions or procedures. Applications that involve Atlas units mated to a supporting plate and that will be soldered to the PCB should take special care to insure that the solder joints are not stressed by the supporting plate once installed. The recommended method to achieve this is to mechanically mate the Atlas to the supporting plate before soldering the Atlas into the PCB. If, for whatever reason, this is not possible, then special care should be taken to insure that the Atlas is precisely aligned with the supporting plate after soldering and before mechanical attachment so that upon mechanical attachment no stress is placed on the Atlas unit, the solder contacts, or the PCB.

*Mounting surface flat and clean.* Thermal performance as well as safe operation of the Atlas requires that the surface that the Atlas is mounted to be flat and clean, free of dust, grease, or foreign objects. The recommended maximum deviation of the mating surface flatness is 3 mils (.076 mm).

*Mechanical mounting limits.* Applications that will utilize a mechanical attachment to the Atlas via the Atlas's mounting tabs should take special care not to overstress the mechanical tabs. Regardless of the attachment method, which is most commonly screws but may also be clips or inserts, the linear force applied to each mechanical tab should not exceed certain values as shown in the following table and the accompanying Figure 2-7.

| Atlas Package Size | Maximum Direct Force Per Tab | Screw Type, Corresponding Maximum Rotary Torque |
|---|---|---|
| Ultra Compact | 25 pounds (111 N) | M2.0 x .40, 11.0oz-in (.078 N-m) |
| Compact | 35 pounds (156 N) | M2.5 x.45, 12.5oz-in (.088 N-m) |

**Figure 2-7: Atlas Torque Specifications**




*Mechanical mounting procedure.* Atlas units that are mated to a heat sink or mechanical plate should be attached by progressively tightening both of the Atlas unit's tabs. This means that one screw may be tightened, followed by the other, than back to the first etc. until the desired torque at each screw has been achieved. Following this procedure is particularly important when installing Atlas units over paste or epoxy, where the subsurface layer will undergo compression and movement before settling to a final installed position.

To ensure that proper contact exists between the Atlas and the entire thermal transfer material substrate, and to ensure that the Atlas unit is not damaged via mechanical overstress, the user should carefully apply equal torque increments to each tab screw, never exceeding at any point the torque limit on either tab of 25 lbs (111 N) linear force or 11.0 oz-in (.078 N-m) rotary torque using a M2.0 x .40 screw for the ultra compact Atlas package, and 35 lbs(156N) linear force or 12.5oz-in (.088 N-m) rotary torque using a M2.5 x .45screw for the compact Atlas package.

.

It is the responsibility of the user to ensure that all Atlas units have been installed within the above prescribed mechanical stress limits and following the above described procedures. Failure to observe any of the above recommended procedures and limits may result in incorrect operation or failure of the Atlas during operation.

# 3. Electrical Specifications

3

***In This Chapter***

- Drive Ratings
- Absolute Maximum Ratings
- Environmental Ratings
- Safety and Compliance
- DC Characteristics
- AC Characteristics
- Pin Descriptions and Pinouts
- Signal Interfacing
- Connection Overview
- Heat Sink Grounding
- Atlas Conversion Factors

## 3.1 Drive Ratings

### 3.1.1 Low Power Units (P/Ns MD2x1048/02xB)

| Specifications* | DC Brush Motor | Brushless DC Motor | Step Motor |
|---|---|---|---|
| Nominal supply voltage (HV) | 12-48 VDC | 12-48 VDC | 12-48 VDC |
| Internal HV capacitance | 33 µF | 33 µF | 33 µF |
| Continuous current | 1.5 ADC | 1.5 Arms | 1.5 Arms |
| Peak current (per phase) | 3.8 A | 3.8 A | 3.8 A |
| Maximum continuous power | 72 W | 88 W | 102 W |

* transformer isolated power supply, T < 40° C

A coldplate or a heatsink in an environment with sufficient airflow can be used to achieve the above drive ratings.

For temperature operation beyond the standard 0-40° C range, above-listed ratings may change. Contact your PMD representative for additional information on Atlas extended temperature operation including higher temperature drive ratings.

### 3.1.2 Medium Power Units (P/Ns MD2x1048/05xB)

| Specifications* | DC Brush Motor | Brushless DC Motor | Step Motor |
|---|---|---|---|
| Nominal supply voltage (HV) | 12-48 VDC | 12-48 VDC | 12-48 VDC |
| Internal HV capacitance | 33 µF | 33 µF | 33 µF |
| Continuous current | 7.0 ADC | 5 Arms | 4.5 Arms |
| Peak current (per phase) | 12.5 A | 12.5 A | 12.5 A |
| Maximum continuous power | 336 W | 294 W | 305 W |

* transformer isolated power supply, T < 40° C

A coldplate or a heatsink in an environment with sufficient airflow can be used to achieve the above drive ratings.

For temperature operation beyond the standard 0-40° C range, above-listed ratings may change. Contact your PMD representative for additional information on Atlas extended temperature operation including higher temperature drive ratings.

### 3.1.3 High Power Units (P/Ns MD2x1056/25xB)

| Specifications* | DC Brush Motor | Brushless DC Motor | Step Motor |
|---|---|---|---|
| Nominal supply voltage (HV) | 12-56 VDC | 12-56 VDC | 12-56 VDC |
| Internal HV capacitance | 33 µF | 33 µF | 33 µF |
| Continuous current | 14.0 ADC | 10.0 Arms | 9.0 Arms |
| Peak current (per phase) | 25.0 A | 25.0 A | 25.0 A |
| Maximum continuous power | 670 W | 590 W | 610 W |

* transformer isolated power supply, T < 40° C

A coldplate or a heatsink in an environment with sufficient airflow can be used to achieve the above drive ratings.

For temperature operation beyond the standard 0-40° C range, above-listed ratings may change. Contact your PMD representative for additional information on Atlas extended temperature operation including higher temperature drive ratings.

## 3.2 Absolute Maximum Ratings

| Parameter | Rating |
|---|---|
| HV voltage range, low power units | 0 V to +52 V |
| HV voltage range, medium power units | 0 V to +52 V |
| HV voltage range, high power units | 0 V to +60 V |
| ~Enable voltage range | -10 V to +24 V |
| SPISI, SPIClk, ~SPICS voltage range | -0.5 V to 6.5 V |
| SPISO voltage range | -0.5 V to 3.7 V |
| FaultOut voltage range | -0.3 V to 24 V |
| FaultOut output current | -35 uA to 50 mA |
| 5V output current, low power units | 50 mA |
| 5V output current, medium power units | 50 mA |
| 5V output current, high power units | 100 mA |

All voltage values are with respect to GND unless otherwise noted.

# 3.3 Environmental Ratings

| Specification | Value |
|---|---|
| Operating ambient temperature | 0 to 40 C |
| Maximum base plate temperature | 75 C |
| Storage temperature | -20 to 85 C |
| Reflow soldering temperature | 300 C (1.5mm for 10 seconds) |
| Humidity | 0 to 95%, non-condensing |
| Altitude | Up to 2,000 meters without derating |
| Contamination | Pollution Degree 2 |

# 3.4 Safety and Compliance

| Specification | Standard |
|---|---|
| CE | LVD: EN60204-1<br>EMC-D: EN61000-6-1, EN61000-6-3, EN55011 |
| Electrical safety | Designed to UL508C, UL840 and EN60204-1 |
| Hazardous materials | RoHS compliant |
| Flammability | UL94-HB |
| Enclosure | IP20 |

# 3.5 DC Characteristics

## 3.5.1 SPISI, SPIClk

| Schmitt-trigger Input | Min | Max | Conditions |
|---|---|---|---|
| $V_+$, Positive-going input threshold voltage | 1.6 V | 2.0 V | |
| V-, Negative-going input threshold voltage | 0.9 V | 1.2 V | |
| VT, Hysteresis V+-V- | 0.6 V | 1.0 V | |
| $I_{IN}$, input current | | ±1 uA | Input voltage is 5.5 V or GND |

## 3.5.2 SPISO

| | Min | Max | Conditions |
|---|---|---|---|
| $V_O$, output voltage | 0 | 3.3 V | |
| $V_{OH}$, Logic 1 output voltage | 3.2 V | | $I_{OH}$=-100 uA |
| | 2.4 V | | $I_{OH}$=-16 mA |
| $V_{OL}$, Logic 0 output voltage | | 0.1 V | $I_{OL}$=100 uA |
| | | 0.7 V | $I_{OL}$=16 mA |

| | Min | Max | Conditions |
|---|---|---|---|
| $I_{OZ}$, input current when ~SPICS is "1" | | 10 uA | $V_O$ = 0 to 3.7 V |

## 3.5.3 ~SPICS

| | Min | TYP | Max | Conditions |
|---|---|---|---|---|
| $V_{IH}$, Logic 1 input voltage | 2 V | | | |
| $V_{IL}$, Logic 0 input voltage | | | 0.8 V | |
| $I_{IN}$, pull-up current | | -500 uA | | |

## 3.5.4 ~Enable

| Schmitt-trigger input | Min | Max | Conditions |
|---|---|---|---|
| $V_+$, Positive-going input threshold voltage | 1.6 V | 2.0 V | |
| V-, Negative-going input threshold voltage | 0.9 V | 1.2 V | |
| VT, Hysteresis V+-V- | 0.6 V | 1.0 V | |

## 3.5.5 FaultOut

| | Min | Max | Conditions |
|---|---|---|---|
| Output impedance with Logic 1 output | 148 Kohm | 152 Kohm | $I_{OH}$=-100 uA |
| $V_{OL}$, Logic 0 output voltage | | 0.25 V | $I_{OL}$=10 mA |

## 3.5.6 5V

| | Min | Max | Conditions |
|---|---|---|---|
| Voltage tolerance, low power units | -5% | 5% | Output current 0-50 mA |
| Voltage tolerance, medium power units | -5% | 5% | Output current 0-50 mA |
| Voltage tolerance, high power units | -5% | 5% | Output current 0-100 mA |
| Short circuit protection | Not available | | |

# 3.6 AC Characteristics

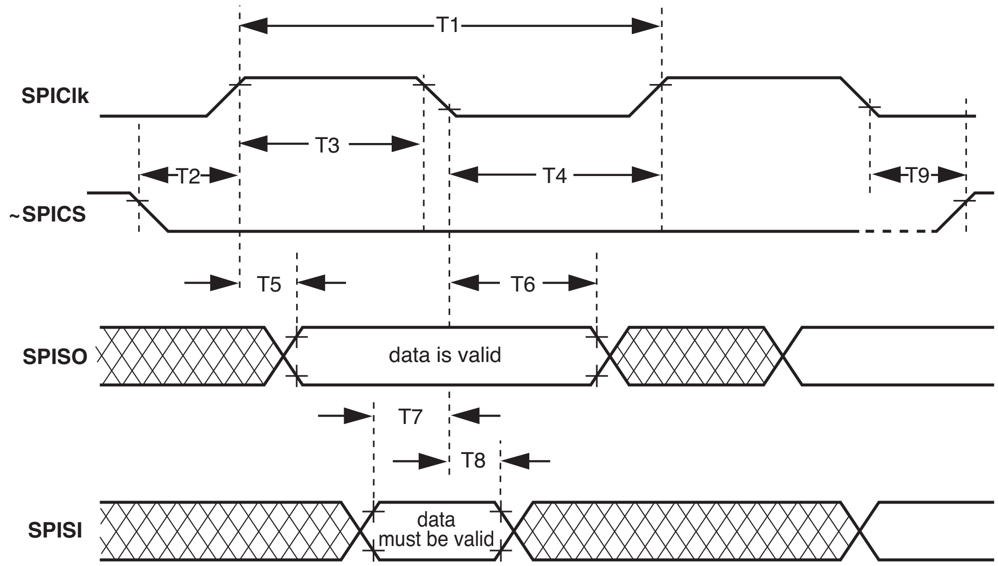



**Figure 3-1: Timing Diagrams**

See Figure 3-1 for timing numbers.

| Timing Interval | No. | Min | Max |
|---|---|---|---|
| $T_{SPI}$, SPI clock cycle time | T1 | 125 nsec | |
| Pulse duration, SPIClk high | T3 | (0.5 $T_{SPI}$-10) nsec | |
| Pulse duration, SPIClk low | T4 | (0.5 $T_{SPI}$-10) nsec | |
| SPIClk high to SPISO valid delay time | T5 | | 30 nsec |
| SPISO date valid time after SPIClk low | T6 | 0 nsec | |
| SPISI setup time before SPIClk low | T7 | 30 nsec | |
| SPISI valid time after SPIClk low | T8 | (0.5 $T_{SPI}$-6) nsec | |
| ~SPICS low to first SPIClk high | T2 | 800 nsec | |
| Last SPIClk low to ~SPICS high | T9 | 0.5 $T_{SPI}$ | |

# 3.7 Pin Descriptions and Pinouts

All Atlas units regardless of package size or mounting configuration provide a common set of signals and functions however the pin addresses and number of pins for various functions are different between the ultra compact Atlas units and the compact Atlas units. In addition, the pin addresses are different between the horizontal and vertical mounting configurations for each package size.

The following sections provide detailed pinouts for the two Atlas packages; ultra compact and compact, and the two mounting configuration; horizontal and vertical.

All Atlas unit pins are 0.1 inch spacing and 0.025inch pin width.

## 3.7.1 Atlas Pinouts - Ultra Compact, Vertical

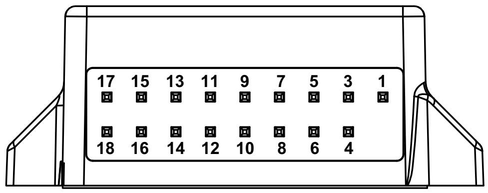



**Figure 3-2: Atlas Pinouts - Ultra Compact, Vertical**

| Pin | Name | Pin | Name |
|---|---|---|---|
| 1 | HV | 2 | |
| 3 | Motor A | 4 | Pwr_Gnd |
| 5 | Motor C | 6 | Motor B |
| 7 | Motor D | 8 | NC (No Connect) |
| 9 | NC (No Connect) | 10 | NC (No Connect) |
| 11 | ~Enable | 12 | FaultOut |
| 13 | GND | 14 | 5V |
| 15 | SPISO | 16 | ~SPICS/AtRest |
| 17 | SPIClk/Pulse | 18 | SPISI/Direction |

The ultra compact Atlas vertical package is keyed so that it is installation direction dependent. It has no physical pin installed at the Pin #2 location.

## 3.7.2 Atlas Pinouts - Ultra Compact, Horizontal

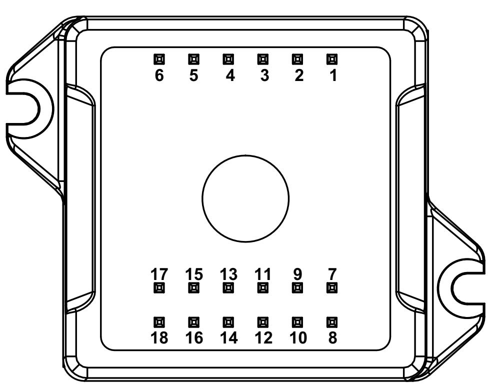



**Figure 3-3: Atlas Pinouts - Ultra Compact, Horizontal**

| Pin | Name | Pin | Name |
|---|---|---|---|
| 1 | Motor D | 2 | Motor C |
| 3 | Motor B | 4 | Motor A |
| 5 | HV | 6 | Pwr_Gnd |
| 7 | SPISI/Direction | 8 | SPIClk/Pulse |
| 9 | SPISO | 10 | ~SPICS/AtRest |

| | | | |
|---|---|---|---|
| 11 | 5V | 12 | GND |
| 13 | FaultOut | 14 | ~Enable |
| 15 | GND | 16 | NC (no connect) |
| 17 | NC (no connect) | 18 | NC (no connect) |

## 3.7.3 Atlas Pinouts - Compact, Vertical

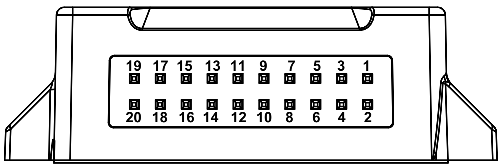



**Figure 3-4: Atlas Pinouts - Compact, Vertical**

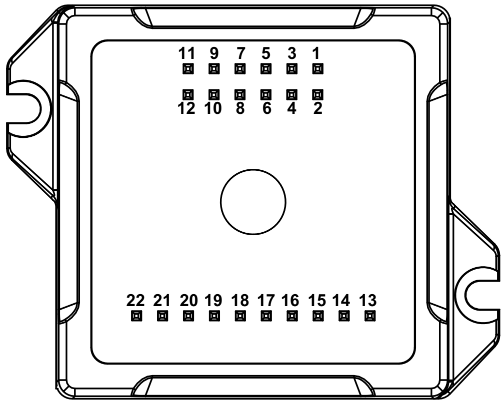
| Pin | Name | Pin | Name |
|---|---|---|---|
| 1 | Pwr_Gnd | 2 | Pwr_Gnd |
| 3 | HV | 4 | HV |
| 5 | Motor A | 6 | Motor A |
| 7 | Motor B | 8 | Motor B |
| 9 | Motor C | 10 | Motor C |
| 11 | Motor D | 12 | Motor D |
| 13 | ~Enable | 14 | FaultOut |
| 15 | 5V | 16 | GND |
| 17 | ~SPICS/AtRest | 18 | SPISI/Direction |
| 19 | SPIClk/Pulse | 20 | SPISO |

The compact Atlas package provides additional power output via doubling of the HV, Pwr_Gnd, and Motor output pins. To achieve the rated unit power output be sure that both pins are connected.

The compact Atlas vertical package is not keyed and therefore care should be taken to install in the correct orientation.

## 3.7.4 Atlas Pinouts - Compact, Horizontal

**Figure 3-5: Atlas Pinouts - Compact, Horizontal**




| Pin | Name | Pin | Name |
|---|---|---|---|
| 1 | Motor D | 2 | Motor D |
| 3 | Motor C | 4 | Motor C |
| 5 | Motor B | 6 | Motor B |
| 7 | Motor A | 8 | Motor A |
| 9 | HV | 10 | HV |
| 11 | Pwr_Gnd | 12 | Pwr_Gnd |
| 13 | 5V | 14 | GND |
| 15 | ~Enable | 16 | FaultOut |
| 17 | GND | 18 | ~SPICS/AtRest |
| 19 | SPISO | 20 | SPISI/Direction |
| 21 | SPIClk/Pulse | 22 | GND |

The compact Atlas package provides additional power output via doubling of the HV, Pwr_Gnd, and Motor output pins. To achieve the rated unit power output be sure that both pins are connected.

## 3.7.5 Pin Descriptions

| Pin Name | Direction | Description |
|---|---|---|
| HV | | DC power to Atlas module, referenced to Pwr_Gnd. The DC power source should be a transformer isolated power supply. For the compact Atlas package two pins carry this signal, so care should be taken to connect both pins. |
| Pwr_Gnd | | Power return for HV, Motor A, Motor B, Motor C and Motor D. For the compact Atlas package two pins carry this signal, so care should be taken to connect both pins. For greatest EMI protection double shielded cables on the motor winding A, B, C, and D should be used with the inner shield connected to Pwr_Gnd and the outer shield connected to chassis ground. |
| Motor A | | Motor output pin A. Used with Brushless DC, DC Brush, and Step Motors. For the compact Atlas package two pins carry this signal, so care should be taken to connect both pins. |
| Motor B | | Motor output pin B. Used with Brushless DC, DC Brush, and Step Motors. For the compact Atlas package two pins carry this signal, so care should be taken to connect both pins. |
| Motor C | | Motor output pin C. Used with Brushless DC, and Step Motors. For the compact Atlas package two pins carry this signal, so care should be taken to connect both pins. |

| Pin Name | Direction | Description |
|---|---|---|
| Motor D | | Motor output pin D. Used with Step Motors. For the compact Atlas package two pins carry this signal, so care should be taken to connect both pins. |
| ~Enable | Input | ~Enable is an active-low input. Should be tied or driven low for Atlas motor output to be active. |
| FaultOut | Output | FaultOut is high impedance when active. It provides programmable fault indication, and is low when inactive. |
| SPIClk/Pulse | Input | SPI input clock or Pulse signal.<br>Pulse is used when Atlas is set to pulse & direction signal mode, and causes a position change command upon a high to low transition. Selection of signal interpretation for this pin is via the SPI communications bus. The default signal interpretation is SPIClk. |
| SPISO | Output | SPI data master in slave out signal. It goes to high impedance when ~SPICS is high. This pin is not used if Atlas is operating in pulse & direction signal mode. |
| SPISI/Direction | Input | SPI data master out slave in signal or Direction signal.<br>Direction is used when Atlas is set to pulse & direction signal mode, and indicates the step direction. Low means the position decreases upon a high to low transition of the Pulse signal, and high means the position increases. Selection of signal interpretation for this pin is via the SPI communications bus. The default signal interpretation is SPISI. |
| ~SPICS/AtRest | Input | ~SPICS signal or AtRest signal.<br>~SPICS enables SPI communication when transitioning low. The SPI block is disabled when ~SPICS is high.<br>*AtRest* is used when Atlas is set to pulse & direction signal mode, and indicates that the step motor holding current should be used rather than the drive current. Selection of signal interpretation for this pin is via the SPI communications bus. The default signal interpretation is ~SPICS. |
| GND | | Ground return for ~Enable, FaultOut, SPI or pulse & direction signals and 5V. |
| 5V | | 5V output used to drive external circuitry. |

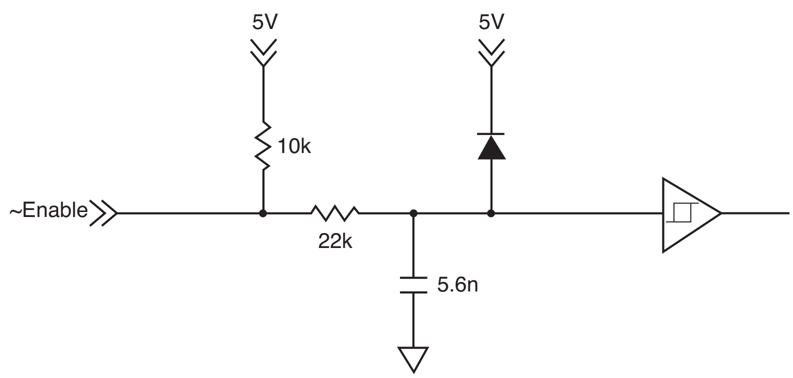
# 3.8 Signal Interfacing

## 3.8.1 ~Enable

~Enable and FaultOut signals are typically used to implement a safety interlock between the Atlas module and other portions of the system.

~Enable is an active low input that must be tied or driven low for the Atlas power output to be active. Its input buffer is shown in Figure 3-6. The circuit accepts signals in the range of 0-24V and has TTL compatible, Schmidt trigger thresholds. It has a pull-up to 5V to allow direct interfacing to open collector enable sources without external pull-up resistor and a 1.3kHz R-C low-pass filter to reject noise.

**Figure 3-6: Signal Interfacing ~ Enable**

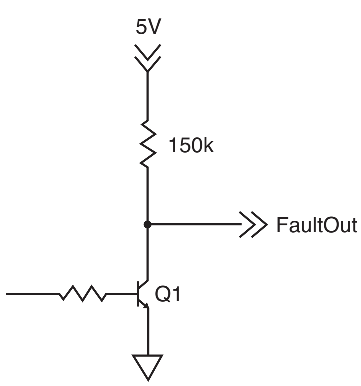



## 3.8.2 FaultOut

FaultOut is asserted high when a fault occurs. The external controller can select which fault conditions drive the *FaultOut* signal.

An Atlas FaultOut output circuit is shown in diagram Figure 3-7. This circuit can continuously sink 50mA when pulled low. It has a 150kohm pull-up resistor to 5V. Its voltage range is 0 to 24V.

**Figure 3-7: Signal Interfacing FaultOut**



# 3.9 Connection Overview

## 3.9.1 Brushless DC Motors

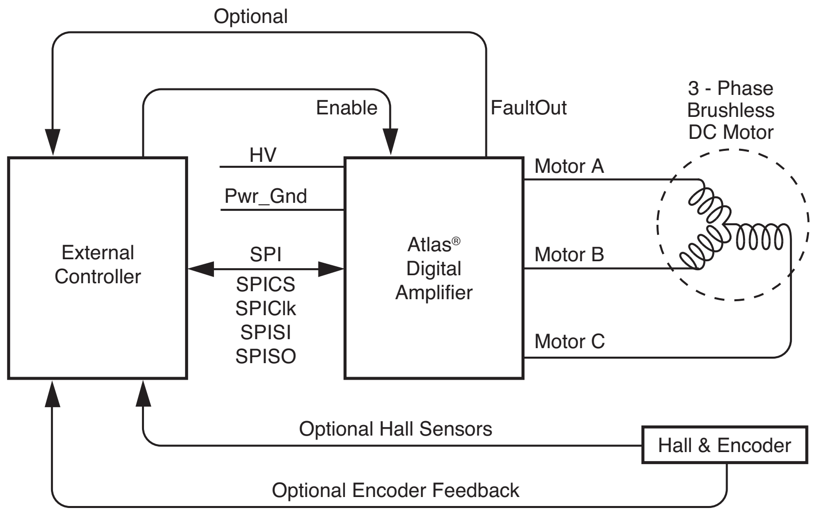



**Figure 3-8: Brushless DC Connections**

The following table summarizes the recommended connections when connecting Atlas amplifiers to Brushless DC motors

| Type | Required Connections | Optional Connections |
|---|---|---|
| Power | HV, Pwr_Gnd | |
| Communication | ~SPICS, SPISO, SPISI, SPIClk, GND | |
| Motor | Motor A, Motor B, Motor C | |
| Miscellaneous | ~Enable | FaultOut |

If Atlas is used as part of a higher level position controller, as shown in the Figure 3-8, the Brushless DC motor provides feedback signals to the external controller. Commonly, both Hall sensor signals and a position encoder are used, but only one or the other is needed in a minimal configuration. In this configuration the external controller generally consists of a PMD Magellan Motion Control IC or a programmable microprocessor or DSP-type device.

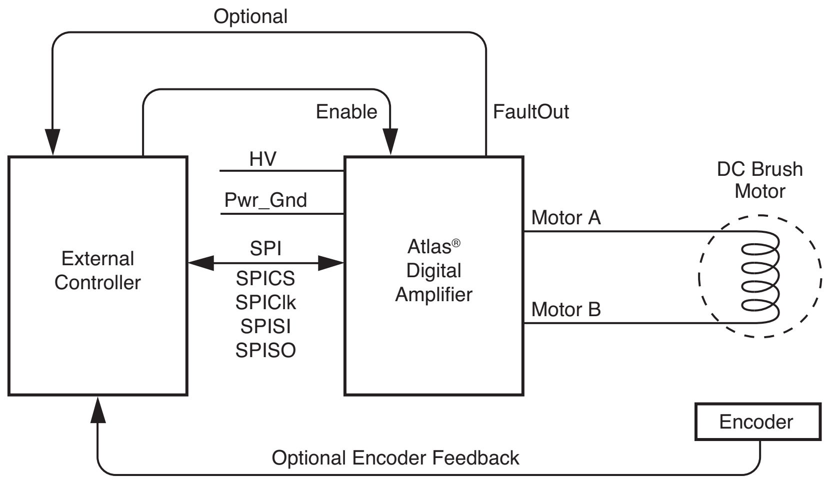
Alternatively, Atlas can be operated by an external controller as a standalone device, driving the motor at commanded voltage or torque levels and not part of a higher-level servo controller. In this configuration, the external controller can be either a microprocessor-type device, or a logic device such as an FPGA (field programmable gate array).

Atlas functions as a power block providing amplification, current control, and safety management of the amplifier and motor. Atlas does not directly accept Hall signals or encoder signals, so to operate with a Brushless DC motor the motor's current phase angle must be provided by the external controller through the SPI interface.

> The Atlas does not support direct Hall signal inputs. To operate the Atlas with a Brushless DC motor, continuous motor phase angle is provided by the external controller, via either Hall inputs or an encoder.

## 3.9.2 DC Brush Motors

**Figure 3-9: DC Brush Connections**




The following table summarizes the recommended connections when connecting Atlas amplifiers to DC Brush motors.

| Type | Required Connections | Optional Connections |
|---|---|---|
| Power | HV, Pwr_Gnd | |
| Communication | ~SPICS, SPISO, SPISI, SPIClk, GND | |
| Motor | Motor A, Motor B | |
| Miscellaneous | ~Enable | FaultOut |

If Atlas is used as part of a higher level servo controller, as shown in Figure 3-9, an encoder provides position or velocity feedback signals to the external controller. In this configuration the external controller generally consists of a PMD Magellan Motion Control IC or a programmable microprocessor or DSP-type device.

Alternatively, Atlas can be operated by an external controller as a standalone device, driving the motor at commanded voltage or torque levels. In this configuration the external controller can be either a microprocessor-type device, or a logic device such as an FPGA (field programmable gate array).

### 3.9.3 Step Motors Using SPI Communications

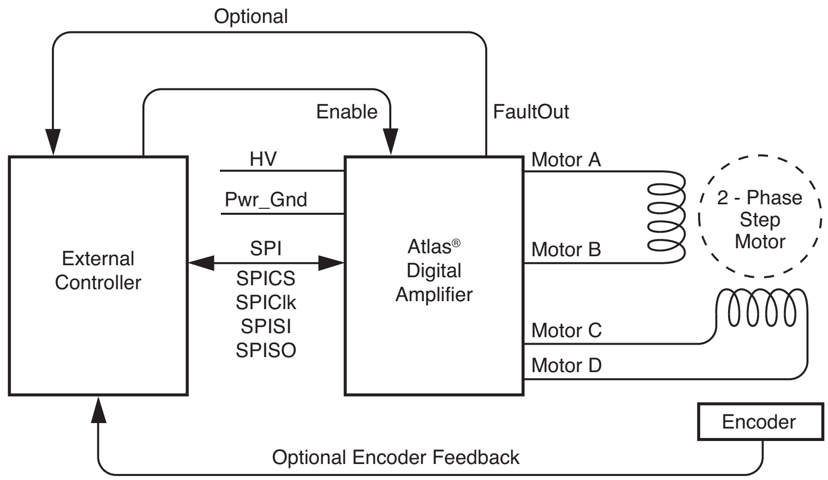



**Figure 3-10: Step Motor SPI Communication Connections**

The following table summarizes the recommended connections when connecting Atlas amplifiers to two-phase step motors when using the SPI communications channel. In this mode the external controller provides position commands to Atlas via the SPI interface. Note that this is the recommended connection method when the Atlas is connected to a Magellan Motion Control IC.

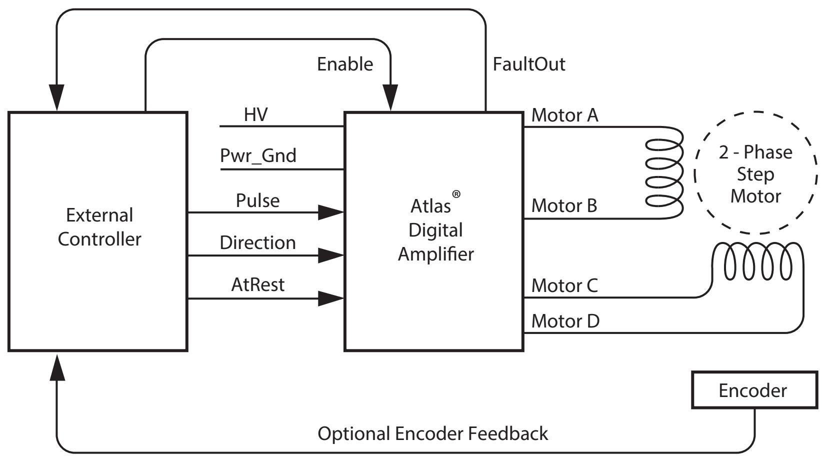
| Type | Required Signal Connections | Optional Signal Connections |
|---|---|---|
| Power | HV, Pwr_Gnd | |
| Communication | ~SPICS, SPISO, SPISI, SPIClk, GND | |
| Motor, Phase $A^+$:<br>Motor, Phase $A^-$<br>Motor, Phase $B^+$:<br>Motor, Phase $B^-$: | Motor A<br>Motor B<br>Motor C<br>Motor D | |
| Miscellaneous | ~Enable | FaultOut |

These connections apply to bipolar motors. If connecting to unipolar motors do not connect the center tap.

In this configuration the external controller generally consists of a PMD Magellan Motion Control IC, a programmable microprocessor or DSP-type device, or a FPGA (field programmable gate array). The external controller provides a continuous stream of position commands to control the motor position.

### 3.9.4 Step Motors in Pulse & Direction Signal Mode

**Figure 3-11: Step Motor Pulse and Direction Mode Connections**




The following table summarizes the recommended connections when connecting Atlas amplifiers to two-phase step motors when using the pulse & direction signal mode. In this mode the external controller provides position commands to Atlas via pulse and direction signals.

| Type | Required Connections | Optional Connections |
|---|---|---|
| Power | HV, Pwr_Gnd | |
| Communication | Pulse, Direction, GND | AtRest |
| Motor, Phase A+: | Motor A | |
| Motor, Phase A- | Motor B | |
| Motor, Phase B+: | Motor C | |
| Motor, Phase B-: | Motor D | |
| Miscellaneous | ~Enable | FaultOut |

These connections apply to bipolar motors. If connecting to unipolar motors do not connect the center tap.

In this configuration the external controller generally consists of an off-the-shelf motion control card or module, a programmable microprocessor or DSP-type device, or a FPGA (field programmable gate array). The external controller provides a continuous stream of pulse and direction commands, along with (optionally) an *AtRest* signal to control the torque.

To initially set up and store its application-specific configuration parameters, Atlas is programmed using the SPI interface and then commanded to convert to pulse & direction signal mode.

*FaultOut* signal input to external controller is strongly recommended when the Atlas is used in Pulse & Direction signal mode.

## 3.10 Heat Sink Grounding

The heat sink may be left ungrounded or may be connected to chassis ground for best EMI protection. The heat sink should not be connected to the Atlas Pwr_Gnd.

# 3.11 Atlas Conversion Factors

The following table provides electrical conversion factors for the various Atlas units. These factors convert Atlas command values specified via the Atlas unit's digital SPI interface (referred to as counts) to physical quantities such as amperage or volts, and vice versa. For more information on the Atlas functions related to these conversion factors see Chapter 4, *Operation*.

| Unit | All Low Power Atlas | All Medium Power Atlas | All High Power Atlas | Example Usage |
|---|---|---|---|---|
| Amps<br>For commands:<br>GetCurrentFoldback<br>SetCurrentFoldback<br>GetFOC<br>SetFOC<br>GetCurrentLoop<br>SetCurrentLoop<br>GetCurrentLoopValue<br>GetFOCValue | .231 mA/count | .763 mA/count | 1.526 mA/count | Note these conversion factors apply to listed commands only. A command request to read the medium power Atlas unit's $I^2t$ continuous current gives a value of 11,335. This corresponds to a current of 11,335 * .763 mA/count = 8.648 amps. |
| Amps<br>For commands:<br>GetCurrent<br>SetCurrent | .116 mA/count | .382 mA/count | .763 mA/count | Note these conversion factors apply to listed commands when current loop is enabled. To command a current of 3.5A to the high power Atlas a value of 3,500 mA / .763 mA/count = 4,587 counts is specified. |
| Volts | 1.361 mV/count | 1.361 mV/count | 1.361 mV/count | A command request to read the Atlas unit's DC Bus voltage gives a value of 12,345. This corresponds to a voltage of 12,345 counts * 1.361 mV/count = 16.8 volts. |
| Temperature | .0039°C/count | .0039°C/count | .0039°C/count | A command request to read the Atlas unit's internal temperature gives a value of 7,890. This corresponds to a temperature of 7,890 counts * .0039°C/count = 30.8°C. |
| Foldback Energy | .0059 $A^2$sec/count | .064 $A^2$sec/count | .256 $A^2$sec/count | To command a foldback energy of 50$A^2$sec to the high power Atlas a value of 50$A^2$sec/.256 $A^2$sec/count = 195 counts is specified. |

## 3.11.1 Atlas Settings Defaults and Limits

The following table provides default and limit values for all Atlas units.

| Quantity | All Low Power Atlas | All Medium Power Atlas | All High Power Atlas |
|---|---|---|---|
| **Overtemperature** | | | |
| Overtemperature Limit | Default: 75.0°C<br>Low Limit: 0 | Default: 75.0°C<br>Low Limit: 0 | Default: 75.0°C<br>Low Limit: 0 |

| Quantity | All Low Power Atlas | All Medium Power Atlas | All High Power Atlas |
|---|---|---|---|
| | High Limit: 75.0°C | High Limit: 75.0°C | High Limit: 75.0°C |
| Overtemperature Hysteresis | Default: 5.0°C<br>Low Limit: 0<br>High Limit: 25.0°C | Default: 5.0°C<br>Low Limit: 0<br>High Limit: 25.0°C | Default: 5.0°C<br>Low Limit: 0<br>High Limit: 25.0°C |
| **Voltage** | | | |
| Overvoltage Limit | Default: 52.0 V<br>Low Limit: 10.0 V<br>High Limit: 52.0 V | Default: 52.0 V<br>Low Limit: 10.0 V<br>High Limit: 52.0 V | Default: 60.0 V<br>Low Limit: 10.0 V<br>High Limit: 60.0 V |
| Undervoltage Limit | Default: 10.0 V<br>Low Limit: 10.0 V<br>High Limit: 48.0 V | Default: 10.0 V<br>Low Limit: 10.0 V<br>High Limit: 48.0 V | Default: 10.0 V<br>Low Limit: 10.0 V<br>High Limit: 56.0 V |
| **Current Foldback** | | | |
| Continuous Current Limit, Brushless DC Motor | Default: 2.12 A<br>Low Limit: 0.0 A<br>High Limit: 2.12 A | Default: 7.07 A<br>Low Limit: 0.0 A<br>High Limit: 7.07 A | Default: 14.1 A<br>Low Limit: 0.0 A<br>High Limit: 14.1 A |
| Continuous Current Limit, DC Brush Motor | Default: 1.50 A<br>Low Limit: 0.0 A<br>High Limit: 1.50 A | Default: 7.00 A<br>Low Limit: 0.0 A<br>High Limit: 7.00 A | Default: 14.0 A<br>Low Limit: 0.0 A<br>High Limit: 14.0 A |
| Continuous Current Limit, Step Motor | Default: 2.12 A<br>Low Limit: 0.0 A<br>High Limit: 2.12 A | Default: 6.36 A<br>Low Limit: 0.0 A<br>High Limit: 6.36 A | Default: 12.7 A<br>Low Limit: 0.0 A<br>High Limit: 12.7 A |
| Energy Limit, Brushless DC Motor | Default: 2.95 $A^2$sec<br>Low Limit: 0.0 $A^2$sec<br>High Limit: 2.95 $A^2$sec | Default: 31.9 $A^2$sec<br>Low Limit: 0.0 $A^2$sec<br>High Limit: 31.9 $A^2$sec | Default: 127.5 $A^2$sec<br>Low Limit: 0.0 $A^2$sec<br>High Limit: 127.5$A^2$sec |
| Energy Limit, DC Brush Motor | Default: 3.63 $A^2$sec<br>Low Limit: 0.0 $A^2$sec<br>High Limit: 3.63 $A^2$sec | Default: 32.2 $A^2$sec<br>Low Limit: 0.0 $A^2$sec<br>High Limit: 32.2 $A^2$sec | Default: 128.7 $A^2$sec<br>Low Limit: 0.0 $A^2$sec<br>High Limit: 128.7 $A^2$sec |
| Energy Limit, Step Motor | Default: 2.95 $A^2$sec<br>Low Limit: 0.0 $A^2$sec<br>High Limit: 2.95 $A^2$sec | Default: 34.7 $A^2$sec<br>Low Limit: 0.0 $A^2$sec<br>High Limit: 34.7 $A^2$sec | Default: 138.9 $A^2$sec<br>Low Limit: 0.0 $A^2$sec<br>High Limit: 138.9 $A^2$sec |

For more information on Atlas overtemperature safety functions see Section 4.8.2, “Overtemperature Fault.”

For more information on Atlas overvoltage and undervoltage safety functions see Section 4.8.4, “Undervoltage Fault.”

For more information on Atlas Current Foldback safety functions see Section 4.8.9, “Current Foldback.”

# 4. Operation

### *In This Chapter*

- Functional Overview
- Internal Block Diagram
- Notes on Command Mnemonics
- Commutation
- Current Loop
- Power Stage
- Status Registers
- Safety Processing Functions
- Step Motor Control
- User Memory Space & Buffers
- Trace Capture
- Power-up
- Non-Volatile (NVRAM) Storage
- Writing and Reading NVRAM Data

## 4.1 Functional Overview

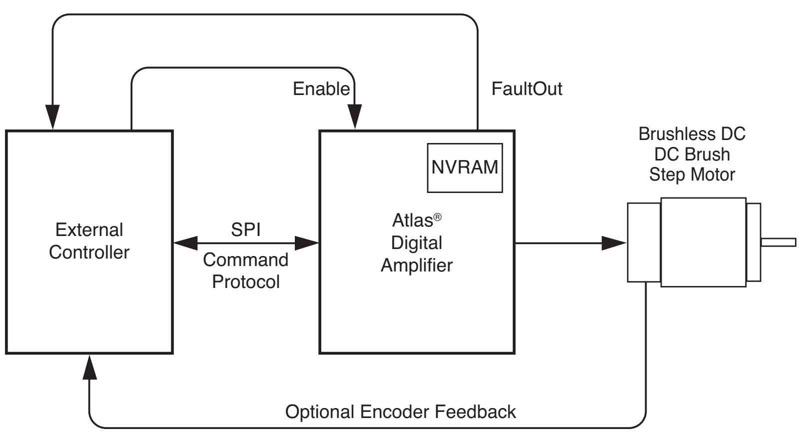



**Figure 4-1: High Level System Diagram**

Atlas Digital Amplifiers are single-axis devices for torque or voltage-mode control of three-phase Brushless DC motors, DC Brush motors, or two-phase step motors. They accept a stream of desired torque or voltage values from an external controller and perform all current loop processing and switching bridge control to continuously drive the motor coils to the specified, commanded values.

In addition to providing a stream of torque or voltage commands, the external controller is used to set up operational parameters needed by Atlas such as control gains, safety-related parameters, and other information. These parameters may be provided to Atlas at each power up, or stored non-volatilely on Atlas so that they no longer need to be loaded at each power-up. See Section 4.12, "Power-up" for more information on non-volatile parameter storage.

Communication to/from Atlas occurs via an SPI interface and associated protocol that uses packet-oriented commands to specify various Atlas parameters, and, if desired, request status information from Atlas. This protocol has been designed for maximum speed and flexibility so that torque or voltage commands can be continuously sent to Atlas even while the external controller queries Atlas for various information. See Chapter 5, *SPI Communications* for more information on the SPI interface.

When Atlas is used in a higher level system such as a servo-based velocity or position controller, torque commands are typically sent to Atlas continuously, at the motion controller's servo rate. For most systems this rate is in the 1,000 to 10,000 samples per second range. However Atlas may also be used with direct voltage or torque control applications that utilize Atlas to specify a desired output value just once after power-up, or only occasionally as required by the application.

To disable Atlas operations it may be powered down, the ***Enable*** signal may be de-asserted, or various commands that result in Atlas operations being suspended may be sent by the external controller to Atlas through the SPI interface. In addition, there are several conditions where Atlas automatically shuts down for safety-related reasons. These may include short circuit detection, under and over voltage protection, $I^2t$ current limiting, and amplifier over temperature detection. See Section 4.8, "Safety Processing Functions" for more information on emergency stop and related functions.

## 4.2 Internal Block Diagram

**Figure 4-2: Internal Block Diagram**

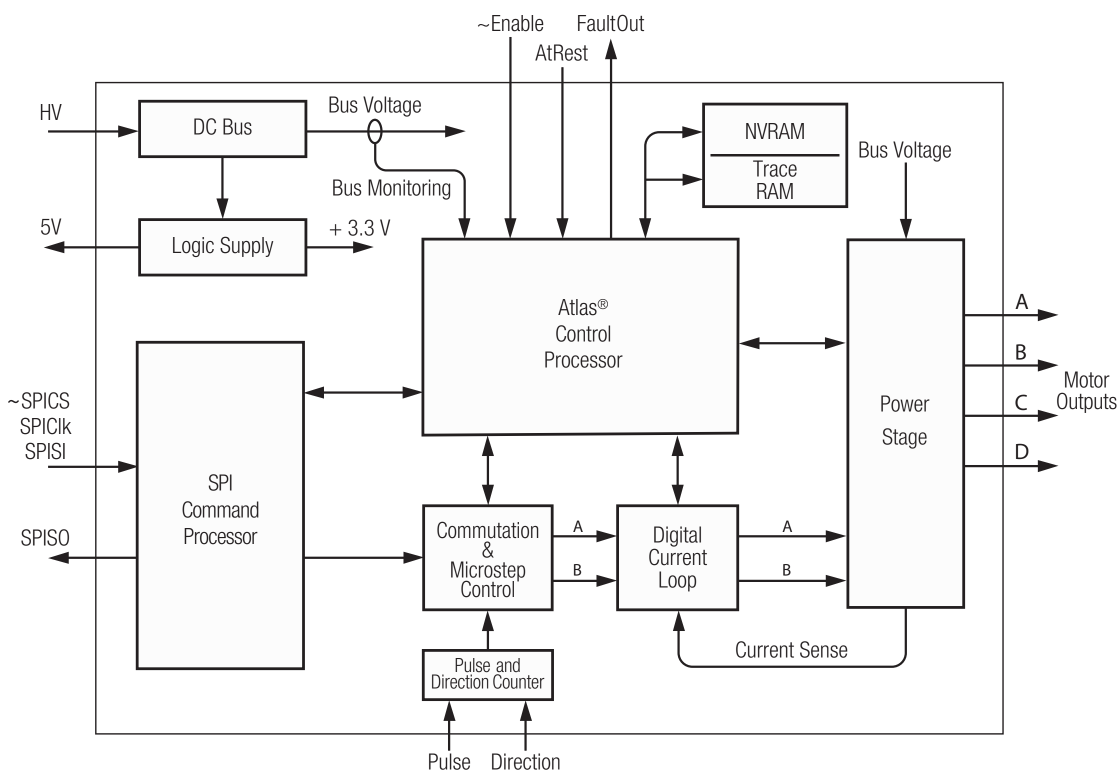



Figure 4-2 shows the internal block diagram of Atlas. Here are summary descriptions of the major modules and functional areas:

**Commutation**—this module utilizes internally generated information, or information provided by the external controller, to split up the desired overall torque command into individual phase commands to drive Brushless DC and step motors.

**Current Loop**—this module inputs the desired current for each motor coil and uses the measured current feedback from each motor coil to develop PWM (pulse width modulation) output command values for the power stage.The current loop module may be disabled, in which case Atlas will drive the motor in voltage mode. See Section 4.5, "Current Loop" for more information on the current loop module.

**Power Stage**—this module receives desired voltages for each motor coil and manages the Atlas unit's high performance MOSFET-based switching bridge to precisely drive the coils of the motor. See Section 4.6, "Power Stage" for a detailed description of this module.

**Status Registers**—this module holds various status registers including an Event Status Register, a Drive Status register, a Drive Fault Status Register, a signal status, and an SPI Status Register.

**Safety Processing**—this module manages Atlas unit safety-related functions including the internal temperature sensor, bus voltage error, the *Enable* input signal, current foldback, the *FaultOut* output signal, event action processing, and more.

**Step Motor Processing**—this module implements step motor-specific features including microstep signal generation, holding current management, and ***Pulse***, ***Direction***, and ***AtRest*** signal processing.

**Memory Buffers**—this module provides user-accessible memory for trace and setup parameter configuration storage.

**Trace**—this module provides a facility for continuously storing up to four simultaneous Atlas variables in the memory buffers.

**Power-up & Non-Volatile Initialization Storage**—this modules manages the power-up sequence and provides the ability to store selected parameter into the Atlas unit's non-volatile memory.

**SPI Command Processor**—This module, described in the next chapter, manages all communications to/from the external controller.

# 4.3 Notes on Command Mnemonics

For simplicity, throughout this and subsequent chapters, we will use a mnemonic-style nomenclature to indicate external controller commands sent to Atlas. Such commands are sent over an SPI (serial peripheral interface) connection, using a packet-based protocol designed to control all aspects of Atlas unit's operation. Atlas supports over 65 commands through this powerful and flexible protocol.

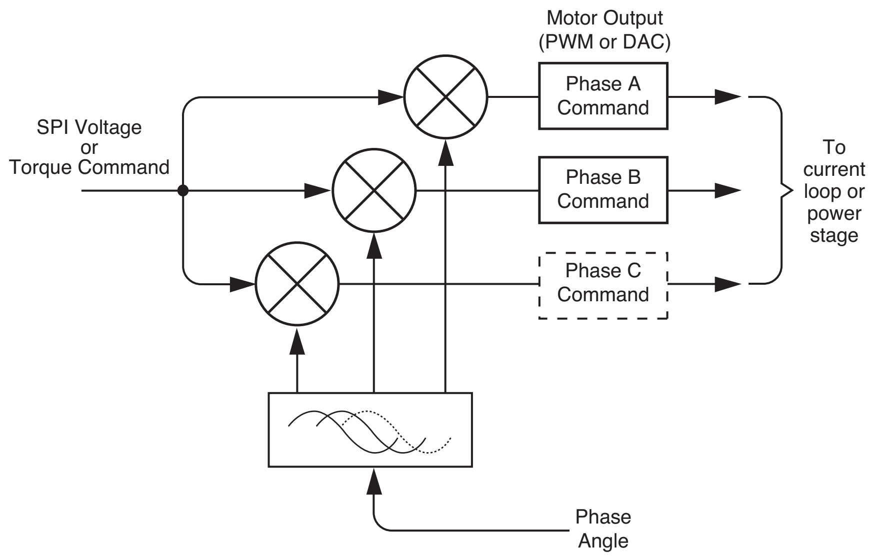
For example, to set the proportional current gain, the command: **SetCurrentLoop** is used.

The actual data packet sent to Atlas to effect the **SetCurrentLoop** command is a specific sequence of signal states on the SPI data lines, not the mnemonic itself. However to more easily illustrate command sequences, the mnemonic format will be used throughout this manual.

See Chapter 5, *SPI Communications*, for complete command details, along with other aspects of the SPI protocol.

# 4.4 Commutation

**Figure 4-3: Commutation Control Sequence**




Brushless DC motors have three phases (generally referred to as A, B, and C) separated from each other by 120 electrical degrees. The process of splitting up the overall torque command into constituent phase commands is called commutation. Figure 4-3 provides an overview of the control sequence when a Brushless DC motor is controlled by Atlas.

The first step is that the external controller specifies the desired motor voltage or torque command to the Atlas. This command is then commutated into constituent phase-specific values. This process applies to step motors as well as Brushless DC motors, however for step motors the process is called microstepping. See Section 4.9, "Step Motor Control" for a detailed discussion of step motor control with Atlas amplifiers. DC Brush motors are single phase devices, and do not require commutation.

Once commutated, the individual commands for the A, B, and C phases are output either directly to the power stage or to the current loop module (depending on whether current control has been requested). If output to the current loop module, additional calculations are performed using the measured current through each winding to determine a final phase command. See Section 4.5, "Current Loop" for details.

## 4.4.1 Determining Phase Angle

Atlas does not directly accept commutation inputs such as Hall sensors, so phase angle information must be provided by the external controller via the SPI interface.

Phase angle in this context means the position of the motor within its overall electrical cycle. Whether or not the electrical phase angle corresponds directly to the mechanical motor angle depends on how many poles the motor has. The most common Brushless DC motor configuration has four poles (two pole pairs), meaning it traverses two full electrical cycles for each rotation of the motor. In this example therefore, the mechanical angle would be half of the electrical angle.

The phase angle that is provided to Atlas by the external controller is encoded as a 12 bit word, consisting of the instantaneous electrical angle of the rotor. A minimum angle of 0 corresponding to an electrical phase angle of 0.0°, and a maximum value of 4,095 corresponding to a value of 359.9°. Phase angles expressed to Atlas are always positive and have a maximum value of 360.0°. This means a motor that moves in the negative direction from a position angle of 0.0 will 'wrap' around to a value of 360.0°. Conversely, a position angle that moves past 360.0° wraps to a value of 0.0°.

To actually send the phase angle to Atlas it is combined with the motor voltage or torque command into a single SPI command packet. Since the phase angle and torque must be provided at each command update cycle, this provides an efficient approach for continuously transmitting Brushless DC motion command packets. See Chapter 5.3, *Sending a Voltage or Torque Output Value* for complete word format and protocol information.

## 4.4.2 Phasing with Hall Sensors

As the previous section indicates, to control a Brushless DC motor the external controller continuously provides phase information to Atlas. Typically, the external controller utilizes either Hall sensors or position encoders to determine this information.

If the external controller directly inputs Halls converting the three incoming hall signal states to a commanded phase angle is straightforward. The table shows how to convert an instantaneous hall sensor reading for the most common Hall encoding scheme to an output command phase angle sent to Atlas over the SPI interface.

| Hall A | Hall B | Hall C | Phase angle to send to Atlas in degrees | 12-bit phase angle word |
|---|---|---|---|---|
| 0 | 0 | 1 | 0° | 0 |
| 1 | 0 | 1 | 60° | 683 |
| 1 | 0 | 0 | 120° | 1,365 |
| 1 | 1 | 0 | 180° | 2,048 |
| 0 | 1 | 0 | 240° | 2,731 |
| 0 | 1 | 1 | 300° | 3,413 |

### 4.4.3 Phasing with an Encoder

**Figure 4-4: Phasing Reference Signals**

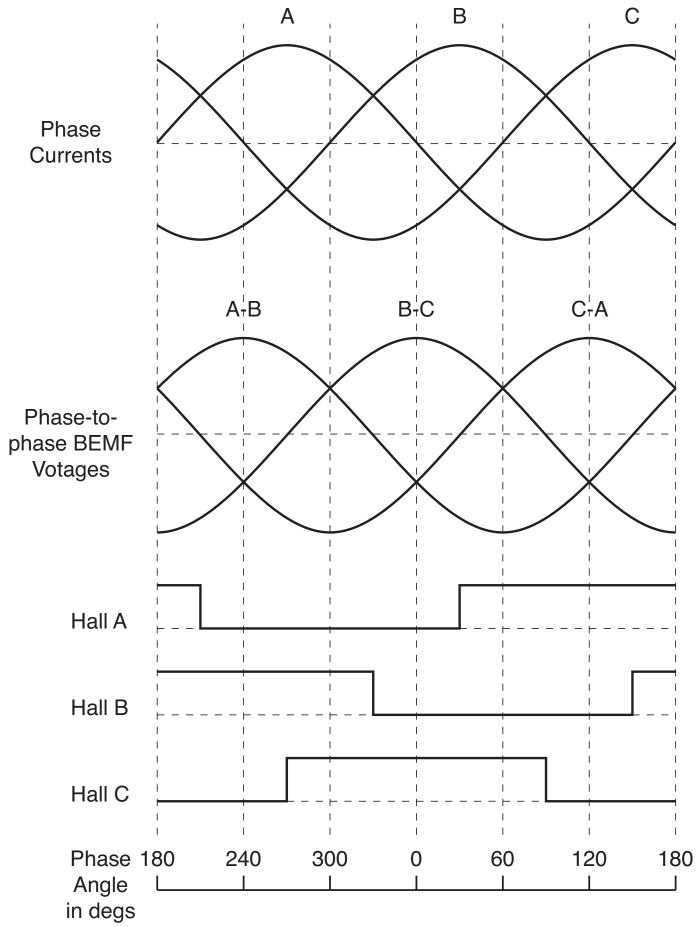



Figure 4-4 shows the relationship between a range of references signals such as Hall signals, and common manufacturer-provided motor control waveforms. Note that these waveforms apply when the motor torque command is positive. If negative, the sign of the "Phase Currents" and "Phase to phase BEMF Voltages" is inverted.

If the external controller uses an encoder to update the phase angle, the phase angle can be sent to Atlas with significantly finer resolution than with Hall sensors (which resolve only to within 60 electrical degrees). Greater commutation resolution allows the motion to be smoother and more efficient. However there are a few important considerations when using an encoder for commutation compared to Hall sensors.

The first is that care should be taken to correctly update the phase angle at the encoder wraparound point. This is the point at which the largest encoder value transitions to the smallest, or vice versa (depending on the motor direction). At these points, the phase angle must still be smoothly and correctly updated as if an encoder wrap had not occurred.

The second is that if an incremental encoder is used and there is the possibility of losing counts, it is important that this be corrected by the external controller. The most common approach toward accomplishing this is to utilize the encoder's Index pulse signal to record a fixed phase angle for the motor, and thereafter compare the incremental encoder reading at each occurrence of the Index pulse. Please refer to the *Magellan Motion Control IC User Guide* for more information on this technique.

The third is known as phase initialization. In the case of incremental encoders, at power-up there is no explicit correlation between the encoder position and the rotor angle, therefore the phasing must be initialized. By contrast, for absolute encoders or certain measuring devices such as resolvers, the phasing may be explicitly known from the encoder angle. Consult the manufacturer's data sheet for more information.

### 4.4.4 Phase Initialization for Incremental Encoders

If an incremental encoder is used to provide continuous phase angle information to Atlas, some method is needed to correlate measured motor angles with phase angle. There are a few commonly used approaches to accomplish this.

The first and simplest is to use Hall sensors. Upon initialization, the table in Section 4.4.2, "Phasing with Hall Sensors" is used to set up the phase angle. As the motor rotates thereafter, the encoder updates the phase angle. For highest accuracy, the phase angle at the transition of one Hall state to another is used. In this manner the initial accuracy, equal to one Hall state width of 60° can be improved to just a few electrical degrees, depending on how accurately the Hall sensors were originally aligned by the motor manufacturer.

If Hall sensors are not available, a technique called algorithmic initialization can be used. In this approach, the Brushless DC motor coils are energized in a specific sequence and the resultant motor reactions are used to determine the initial phase angle. Particularly for free-wheeling motors such as spindles, centrifuges, fans, and similar devices, this approach can work well. However a detailed discussion of this is beyond the scope of this manual, so consult your PMD representative for more information.

## 4.5 Current Loop

> The next section describes a number of concepts that apply even when the current loop is not enabled. All Atlas users should therefore read this section, whether they plan to operate the Atlas with a current loop, or without a current loop in voltage mode.

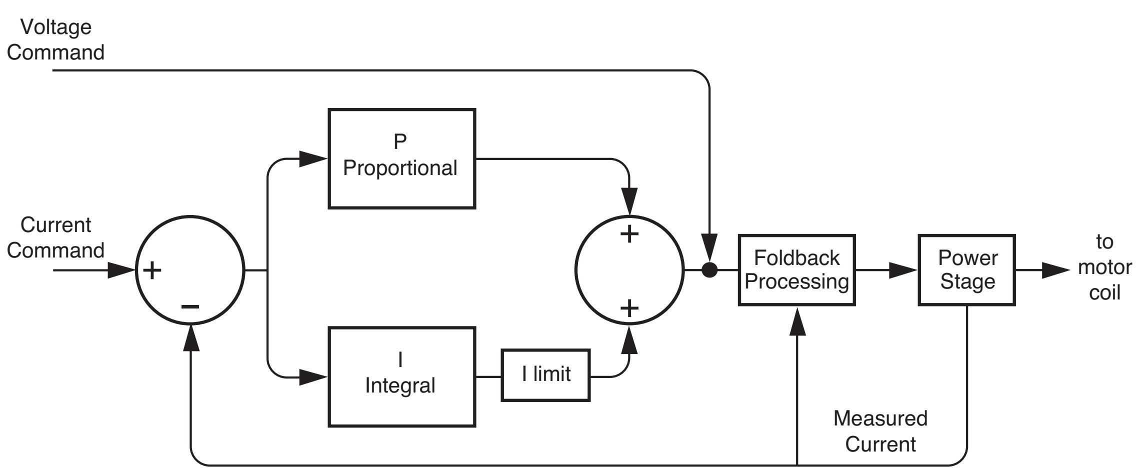



**Figure 4-5: Current Loop Control Flow**

Digital current control is a technique used with DC Brush, Brushless DC, and step motors for precisely controlling the current through each winding of the motor. By controlling the current, response times are improved and motor efficiency is increased.

Figure 4-5 provides an overview of Atlas unit's current controller. For single-phase motors such as DC Brush, one current loop per axis is used. For Brushless DC motors, two current loops are used and the third phase command is derived from the other two phases. When driving step motors, two current loops are used, one for the phase A coil, and one for the phase B coil.

There are three overall methods of current control provided by Atlas, however not all methods are used with all motor types. The table below summarizes which current control modes are available with the three motor types supported by Atlas, along with the default configuration for that motor type.

| Current Control Method | Brushless DC | DC Brush | Step Motor |
|---|---|---|---|
| A/B Control | ✓ | ✓ (default) | ✓ (default) |
| Field Oriented Control | ✓ (default) | | ✓ |
| Third Leg Floating | ✓ | | |

To drive Brushless DC motors, the large majority of applications will use FOC (Field Oriented Control). FOC usually provides the highest top speeds and more energy efficient operation of the motor compared to A/B current control. Third leg floating is an option that is only available for Hall-commutated motors. In that configuration, third leg floating can sometimes provide a higher top speed than FOC.

Two-phase step motors should always use A/B control mode, whether operated in microstepping mode or in closed loop stepper mode. Although FOC is an allowed control mode for two-phase step motors it is generally not recommended.

DC Brush motors are controlled using A/B, which in this case means a single current loop drives the DC Brush motor's single coil.

If during normal operation the current loop is disabled, then the output from the commutation module will use the "voltage command" pathway shown in Figure 4-5, with no PI current loop. 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.

## 4.5.1 Enabling and Disabling the Current Loop

If during normal operation the current loop is disabled, then the output from the commutation module 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.

> Even when operated in the voltage mode, the user must still select the current control method. This is because selection of this control method also affects aspects of the power stage, specifically the use of space vector PWM, versus sinusoidal PWM, versus standard single-phase PWM generation.

To disable the motor output 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 4.8.9, “Current Foldback” 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.

## 4.5.2 A/B Current Control

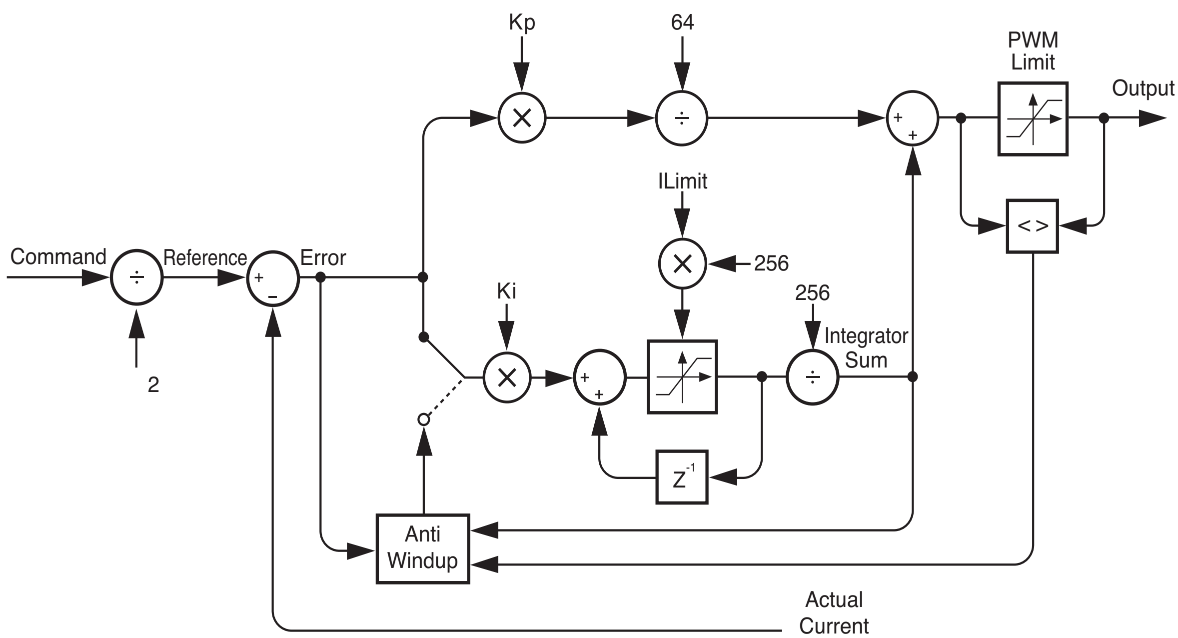



**Figure 4-6: A/B Current Control Calculation Flow**

When A/B current control mode is selected Atlas utilizes the commanded current for each motor winding provided by the commutation module, along with the actual measured current provided by circuitry within the power stage, to perform current loop calculations.

As can be seen in Figure 4-6, the desired current and measured current are 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 power stage module to generate precise PWM (pulse width modulation) output signals, representing the applied voltage, that control the power stage’s switching bridge.

To enable A/B current control the command **SetCurrentControlMode** is used. The value set using this command can be read back using **GetCurrentControlMode**.

Three parameters are set by the user to control the current loop; *Kp*, *Ki*, and *Ilimit*. Two of these are gain factors for the PI controller, and the other is a limit for the integral contribution.

To set any of these three parameters the command **SetCurrentLoop** is used. To read back these parameters, the command **GetCurrentLoop** is used. For multi-phase motors, the 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. The values set using this command are buffered, meaning they are held by Atlas in a buffer but do not instantaneously become active. Buffered commands are activated using the SPI Header. See Section 5.2, “Packet Header” for details.

> It is the responsibility of the user to determine control parameters that are suitable for use in a given application.

### 4.5.2.1 Reading Current Loop Values

To facilitate tuning there are a number of current loop values that can be read back as well as traced. To read back these values the command **GetCurrentLoopValue** is used. See Section 4.11.1.3, “Trace Variables” to specify these values for trace during automatic trace capture.

The variables within the current loop that can be read or traced when the control loop mode is set to A/B control are summarized in the following table. Refer to Figure 4-6 when viewing this table.

| Variable Name | Function |
|---|---|
| Phase A Reference, Phase B Reference | *Brushless DC & microstepping motor:*<br>These registers hold the commanded (reference) currents for the phase A and phase B coils.<br>*DC Brush motor:*<br>Phase A Current holds the commanded (reference) current for the motor. |
| Phase A Current, Phase B Current, Phase C Current | *Brushless DC:*<br>These registers hold the measured actual currents for the phase A, phase B, and phase C coils.<br>*Step motor:*<br>These registers hold the measured currents for the phase A, and phase B coils.<br>*DC Brush motor:*<br>Phase A Current holds the measured current for the motor. |
| Phase A Error, Phase B Error | *Brushless DC & microstepping motor:*<br>These registers hold the difference between the current loop reference and the measured current value (Phase A Current, Phase B Current).<br>*DC Brush motor:*<br>The Phase A Error register holds the difference between the current reference and the measured current value (Phase A Current). |
| Phase A Integrator Sum, Phase B Integrator Sum | *Brushless DC & microstepping motor:*<br>These registers hold the sum of the integrator for the phase A and B current loops.<br>*DC Brush motor:*<br>Phase A Integrator Sum holds the sum of the integrator for the current loop |
| Phase A Output, Phase B Output | *Brushless DC & microstepping motor:*<br>These registers hold the output command for the phase A and B current loop.<br>*DC Brush motor:*<br>Phase A Output holds the output command for the current loop |

### 4.5.2.2 A/B Current Control with Step Motors

The Atlas unit’s A/B current control mode is designed to work with both 3-phase Brushless DC motors and 2-phase step motors. When operating step motors in this mode (see Section 4.9, “Step Motor Control” for more information on operations with step motors), the basic method is identical. The same three current loop parameters described in Section 4.5.2, “A/B Current Control” are set, and the readable parameters are also the same.

### 4.5.2.3 A/B Current Control in Voltage Mode

If Atlas is operated in A/B current control mode with the current loop disabled, then after commutation (Brushless DC motors) or microstep signal generation (step motors) the phase-specific commands are output directly to the power stage with no current loop performed.

For example, if the incoming torque command provided by the external controller is 25% full scale, then for an Atlas that is operating with a bus voltage of 24V, the average voltage presented at the motor coil will be 25% of 24V or 6V. For single phase motors such as DC Brush, the PWM generator directly outputs this external controller-commanded value to the power stage. For multi-phase motors such as Brushless DC or step motor, the PWM generator outputs this commanded value after commutation (Brushless DC motors) or microstep signal generation (step motors) to the power stage.

See Section 4.6, "Power Stage" for more information on power stage operations.

## 4.5.3 Field Oriented Control

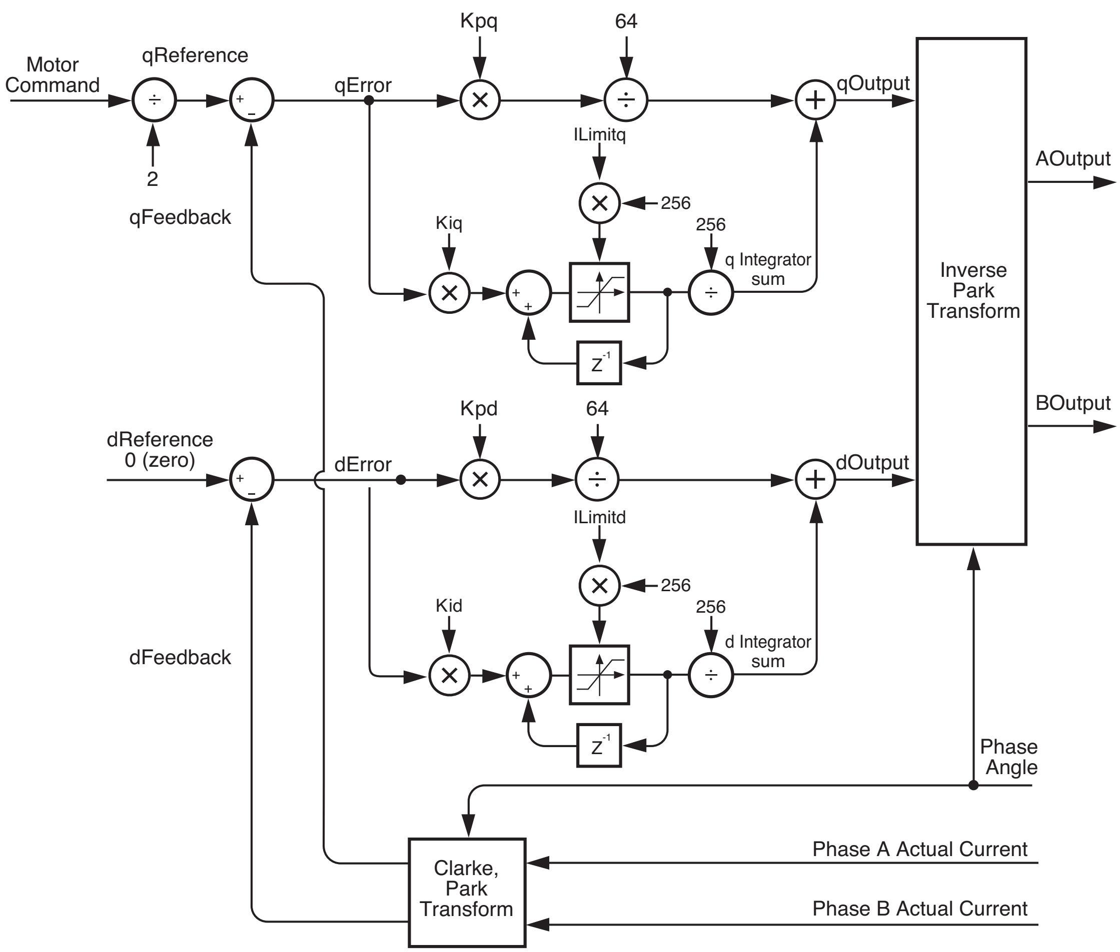



**Figure 4-7: Field Oriented Control Calculation Flow**

Figure 4-7 provides an overview of the calculation flow when field oriented control (FOC), which should only be used to control Brushless DC motors, is selected. Instead of separating phases as A/B control mode does, FOC combines them and "re-references" them to what are known as d (direct torque) and q (quadrature torque) reference frames.

To enable field oriented control mode the command **SetCurrentControlMode** is used. The value set using this command can be read back using **GetCurrentControlMode**.

For each control loop (d and q) three parameters are set by the user, Kp, Ki, and I limit. Two of these are gain factors for the PI (proportional, integral) controller that comprises the heart of the FOC controller, and the other is a limit for the integral contribution.

To set these parameters the command **SetFOC** is used. To read back these parameters the command **GetFOC** is used. The values set using this command are buffered and are activated using the SPI header. See Section 5.2, "Packet Header" for details.

> It is the responsibility of the user to determine control parameters that are suitable for use in a given application.

### 4.5.3.1 Reading FOC Loop Values

To facilitate tuning there are a number of FOC loop values that can be read back as well as traced. To read back these values the command **GetFOCValue** is used. See Section 4.11, "Trace Capture" to specify these values during automatic trace capture.

Refer to Figure 4-7 for an overview of the FOC loop. The variables within the FOC loop that can be read or traced are summarized as follows:

| Variable Name | Function |
|---|---|
| q Reference, d Reference | Are the commanded values input into the q and d loops. Note that d is always set to 0 (zero). |
| q Feedback, d Feedback | Are the measured values for the q (quadrature) and d (direct) force after re-referencing from the actual measured current in the phase A, phase B coils. |
| q Error, d Error | Are the differences, for the q loop and the d loop, between the loop reference and the loop measured value. |
| q Integrator Sum, d Integrator Sum | Are the integrator sums for the d and q loops. |
| q Output, d Output | Are the output commands of the q and the d loops. |
| FOC $\alpha$ Output, FOC $\beta$ Output | Are the FOC outputs in the $\alpha$, $\beta$ reference frame. |
| Phase A Actual Current, Phase B Actual Current | Are the measured currents for the phase A and phase B coils. |

### 4.5.3.2 FOC in Voltage Mode

If Atlas is operated in FOC mode with the current loop disabled, then after commutation the phase-specific commands are output directly to the power stage with no current loop performed.

However unlike the independent phase control mode, a space vector modulation scheme is used to generate the PWM signals and control the switching bridge. Space vector modulation is recommended for most applications because it provides a larger effective range of voltage drive capacity.

See Section 4.6, "Power Stage" for more information on power stage operations.

## 4.5.4 Third Leg Floating Control

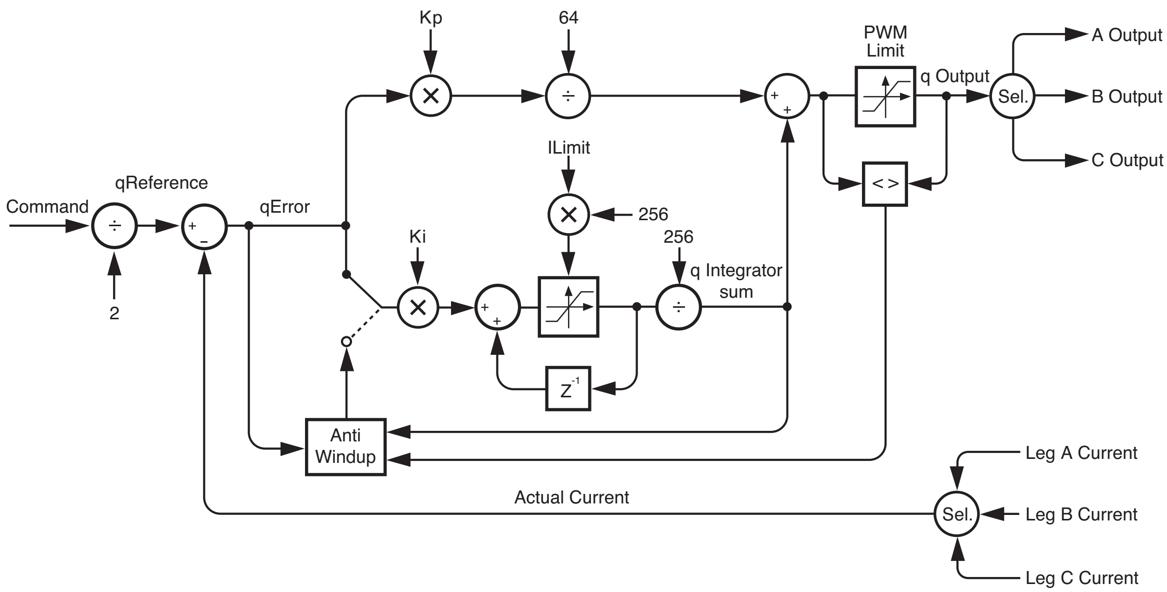



**Figure 4-8: Third Leg Floating Control**

Figure 4-8 provides an overview of the calculation flow when third leg floating control mode is selected. Compared to A/B control or FOC, third leg floating uses a different method in that only two of three legs are driven at any instant with the third, non-driven, leg floating. The actual driven and non-driven legs continuously change based on the phase, as does the leg current used as input to the current loop. In this way, as the motor rotates, each motor leg will go through a sequence of being driven for two cycles and then left floating for one.

To enable third leg floating mode the command **SetCurrentControlMode** is used. The value set using this command can be read back using **GetCurrentControlMode**.

Other than the method by which the motor phases are driven and the leg current is sensed, third leg floating is similar to FOC, however with only the q loop calculated. For the q current loop three parameters are set by the user, Kp, Ki, and Ilimit. Two of these are gain factors for the PI (proportional, integral) controller that comprises the heart of the third leg floating controller, and the other is a limit for the integral contribution. To set any of these parameters the command **SetFOC** is used. To read back these parameters the command **GetFOC** is used. The values set using this command are buffered and may be activated using the update bit of the SPI header. See Section 5.2, “Packet Header” for more information on command buffering.

> It is the responsibility of the user to determine control parameters that are suitable for use in a given application.

> The third leg floating control mode is applicable to Brushless DC motors only.

#### 4.5.4.1 Reading Third Leg Floating Loop Values

To facilitate tuning there are a number of third leg floating loop values that can be read back as well as traced. To read back these values the command **GetFOCValue** is used. See Section 4.11.1.3, "Trace Variables" to specify these values for trace.

Refer to the diagram in Section 4.5.4, "Third Leg Floating Control" for an overview of the control loop. The variables that can be read or traced are summarized as follows:

| Variable Name | Function |
|---|---|
| q Reference | Is the commanded value input into the q loop. |
| q Feedback | Is the measured value of the q component of the current |
| q Error | Is the difference between the measured q component of the current and the commanded q component. |
| q Integrator sum | Is the integrator sum for the q loop. |
| q Output | Is the output command of the q loop |
| q Actual Current | Is the measured current for the q current |

#### 4.5.4.2 Third Leg Floating in Voltage Mode

If Atlas is operated in third leg floating mode with the current loop disabled then the external controller-provided torque command is used to specify PWM duty cycle (voltage) to two out of the three motor output terminals, one positive and one negative. The third terminal is put into a high impedance (floating) state. Which terminal is positive, negative, or floating depends on the controller-provided phase angle.

## 4.6 Power Stage

Atlas contains a high performance MOSFET-based power stage that utilizes one or more switching bridges to drive the motor coils. The use of 3-phase and H-bridge topologies provides full 4-quadrant operation for all motor types. In addition, Atlas uses an advanced PWM switching scheme that minimizes the ripple current on the motor windings while maximizing the current loop performance. The fundamental frequency of the ripple current is twice the PWM frequency, and well out of the audible range in all cases.

The PWM frequency is selectable between 20 kHz, 40kHz, 80kHz, and 120kHz to cover a broad range of motor inductances.

In addition to the output bridge function, Atlas provides a current measurement function for use by the current loop module as well as by the safety processing module. Two channels of phase current feedback are provided for Brushless DC and step motor current loops. For DC Brush motors feedback for one phase is provided.

To set the Atlas unit's PWM rate the command **SetPWMFrequency** is used. To read this value, the command **GetPWMFrequency** is used.

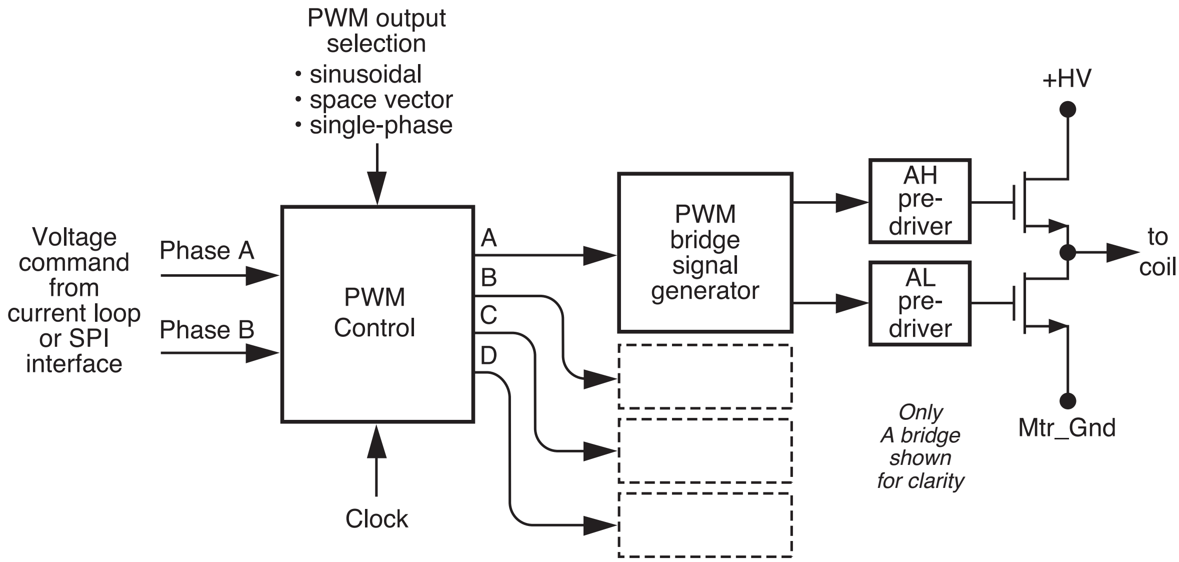



**Figure 4-9: Power Stage Control Flow**

In addition to the output bridge function, Atlas provides a current measurement function for use by the current loop module as well as by the safety processing module. Two channels of phase current feedback are provided for Brushless DC and step motor current loops. For DC Brush motors feedback for one phase is provided.

Three selectable control methods are provided; independent phase, field oriented control, and third leg floating. The choice of control method affects the power stage in the selection of the PWM generation technique. The table below shows this.

| Control Mode | PWM Output Method |
|---|---|
| Independent Phase | sinusoidal |
| Field Oriented Control | space vector modulation |
| Third Leg Floating | standard single-phase |

To select the control method use the **SetCurrentControlMode** command. To read the value set using this command use **GetCurrentControlMode**.

## 4.6.1 PWM Output Limiting

In some applications it may be desirable to limit the maximum allowed output of the power stage PWM generator. For example if the bus voltage is 36 volts, and the desired voltage limit for a particular motor is 18 volts, a PWM limit of 50% is programmed.

Depending on the Atlas unit bus voltage and the effective inductance of the system being controlled, under some circumstances lowering the maximum PWM duty cycle may not fully limit the effective voltage experienced by the device. If this is the case for your system, you may consider increasing the Atlas unit PWM frequency, adding an inductor to the motor circuit, or consulting a PMD representative for more information.

To set the PWM limit value the command **SetDrivePWM** is used. To read this value back the command **GetDrivePWM** is used.

> The programmed drive limit value affects the PWM duty cycle only. It does not limit the effective current that is delivered to the motor. To explicitly limit the current, the current foldback mechanism can be used. See Section 4.8.9, "Current Foldback" for more information.

## 4.6.2 Disabling the Power Stage

During normal operation the Atlas unit's primary function is to drive the motor at the torque or voltage requested by the external controller. However there are a number of circumstances where it may be desirable to disable the power stage. In particular, the power stage may be disabled if certain safety-related conditions occur, or for system calibration. See Section 4.8, "Safety Processing Functions" for more information on Atlas safety processing.

If the power stage module is disabled, all external controller-provided voltage or torque commands are ignored, and all bridge FETs are turned off. This 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 a gravitational load) depending on the load, inertia, and configuration of the axis mechanics.

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

The default condition of the power stage is disabled. Therefore to begin drive operations, a **SetOperatingMode** command must be sent to enable the drive stage module.

## 4.6.3 Enabling the Power Stage

A previously disabled power stage 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 this module was disabled as part of an automatic safety event-related action (see Section 4.8.9.3, "Current Foldback Event Processing" for more information), then the command **RestoreOperatingMode** is used.

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

Regardless of how the module is re-enabled, at the time that the re-enable operation is requested, the power stage module will begin normal operations within approximately 1.0 milliseconds. Care should therefore be taken to re-enable the power stage when the motor axis is in a stable condition such that no abrupt motion occurs.

> If Atlas is in a condition where the current loop module also needs to be re-enabled, both the power stage and the current loop module should be enabled at the same time. This is normally the case when recovering from all safety processing conditions. See Section 4.8, "Safety Processing Functions" for more information.

> It is the responsibility of the user to manage the operation of the power stage so that appropriate safety conditions are maintained at all times.

# 4.7 Status Registers

In addition to various numerical registers that may be queried by the external controller, there are five bit-oriented status registers.

These status registers conveniently combine a number of separate bit-oriented fields into a single register. These registers are Event Status, Drive Status, Signal Status, SPI Status, and Drive Fault Status Register. The external controller may directly query these four registers, or the contents of these registers may be utilized by other functional portions of Atlas, such as *FaultOut* signal processing. See Section 4.8.8, "FaultOut Signal" for more information on *FaultOut* processing.

## 4.7.1 Event Status Register

The Event Status register is defined in the following table:

| Bit | Name | Description |
|---|---|---|
| 0-6 | Reserved | May contain 0 or 1. |
| 7 | Instruction error | Set when an instruction error occurs. This may be due to an incorrect command transmission, an Atlas-detected checksum error, or various other SPI interface command-related conditions. |
| 8 | Reserved | May contain 0 or 1. |
| 9 | Overtemperature fault | Set when an overtemperature fault occurs. |
| 10 | Drive exception | Set when a drive exception event such as bus voltage fault, watchdog timer fault, overcurrent fault, or deassertion of the enable pin occurs that causes Atlas to disable output. |
| 11 | Reserved | May contain 0 or 1. |
| 12 | Current foldback | Set when current foldback occurs. |
| 13-15 | 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 an external controller instruction or a 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.

## 4.7.2 Drive Status Register

The Drive Status register is different than the Event Status register in that the contents are not latched, but rather continuously set and reset by Atlas 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 | Reserved | May contain 0 or 1. |
| 1 | In foldback | Set 1 when in foldback, cleared 0 if not in foldback. |
| 2 | Overtemperature | Set 1 when currently in an overtemperature condition. Cleared 0 if currently not in an overtemperature condition. |
| 3 | Reserved | May contain 0 or 1. |
| 4 | In holding | Set 1 when in a holding current condition. Cleared 0 if not in a holding current condition. |
| 5 | Overvoltage | Set 1 when currently in an overvoltage condition. Cleared 0 if currently not in an overvoltage condition. |

| Bit | Name | Description |
|---|---|---|
| 6 | Undervoltage | Set 1 when currently in an undervoltage condition. Cleared 0 if currently not in an undervoltage condition. |
| 7 | Disabled | Set 1 when the Atlas unit's *Enable* pin is inactive. Cleared 0 when the Atlas unit's *Enable* pin is active. |
| 8-11 | Reserved | May contain 0 or 1. |
| 12 | Output clipped | Set 1 when current loop output clipping is occurring. This occurs when the PWM limit value becomes the output of the current loop due to the fact that the desired output value of the current loop exceeds the PWM Limit value. Cleared 0 when output clipping is not occurring. |
| 13-14 | Reserved | May contain 0 or 1. |
| 15 | Drive not initialized | Upon powerup this bit is set to 1. Cleared 0 when Atlas has finished its power-up initialization sequence and is ready for normal SPI command processing. |

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

### 4.7.3 Signal Status Register

The Signal Status register provides real-time signal levels for some Atlas hardware signals. The Signal Status register is defined in the following table:

| Pin | Name | Description |
|---|---|---|
| 0-12 | Reserved | May contain 0 or 1. |
| 13 | *Enable* pin | Set 1 when a high voltage is present at the *Enable* pin (enabled). Cleared 0 when there is a low signal (disabled). |
| 14 | *FaultOut* pin | Set 1 when a high voltage is being output at the *FaultOut* pin (fault condition present). Cleared 0 when there a low is being output (fault condition not present). |

The command **GetSignalStatus** returns the contents of the Signal Status register.

### 4.7.4 SPI Status Register

The SPI Status Register is used during SPI communications to provide a quick summary status of the overall Atlas amplifier. See Chapter 5, *SPI Communications* for more information on SPI communications processing.

There is no way to set or clear the bits in this register. The bits in this register simply echo an amalgam of bits in various other status registers, as noted in the table below. To change the value of these bits it is therefore necessary to utilize the commands that are associated with those particular status registers.

The SPI Status register is defined in the following table:

| Bit | Name | Description |
|---|---|---|
| 0 | In foldback | Echoes the in foldback bit of the Drive Status register. Set 1 when in foldback, cleared 0 if not in foldback |
| 1 | Overtemperature | Echoes the overtemperature bit of the Drive Status register. Set 1 when currently in an overtemperature condition. Cleared 0 if currently not in an overtemperature condition. |
| 2-3 | Reserved | May contain 0 or 1. |
| 4 | Overvoltage | Echoes the overvoltage bit of the Drive Status register. Set 1 when currently in an overvoltage condition. Cleared 0 if currently not in an overvoltage condition. |
| 5 | Undervoltage | Echoes the undervoltage bit of the Drive Status register. Set 1 when currently in an undervoltage condition. Cleared 0 if currently not in an undervoltage condition. |

| Bit | Name | Description |
|---|---|---|
| 6 | Disabled | Echoes the Disabled bit of the Drive Status register. Set 1 when the *Enable* pin is inactive. Cleared 0 when the *Enable* pin is active. |
| 7 | Instruction error | Echoes the Instruction error bit of the Event Status register. Set 1 when an instruction error occurs. This may be due to an incorrect command transmission, an Atlas-detected checksum error, or various other SPI interface command-related conditions. |
| 8 | Reserved | May contain 0 or 1 |
| 9 | Over temperature event | Echoes the overtemperature bit of the Event Status register. Set 1 when an overtemperature fault occurs. |
| 10 | Drive exception event | Echoes the drive exception bit of the Event Status register. Set 1 when a drive exception event such as bus voltage fault, watchdog timer fault, over current fault, occurs that causes Atlas to disable output. |
| 11 | Output clipped | Echoes the output clipped bit of the Drive Status register. Set 1 when current loop output clipping is occurring. Cleared 0 when output clipping is not occurring. |
| 12 | Foldback event | Echoes the foldback bit of the Event Status register. Set 1 when current foldback occurs. |
| 13-15 | Reserved | May contain 0 or 1. |

The SPI Status Register is not directly read via an Atlas command. It is read as part of SPI header processing. See Section 5.2.1, "Header Return Words" for more information. In addition, it may be selected as a traceable variable. See Section 4.11, "Trace Capture" for more information on tracing.

# 4.8 Safety Processing Functions

Atlas provides a number of amplifier control features that automatically detect and manage safety-related conditions. In addition, Atlas can signal when various conditions, safety or otherwise, occur.

The subsequent sections describe these features.

## 4.8.1 Overcurrent Fault

Atlas supports automatic detection of excessive current output. This fault occurs when the motor, the wiring leading from Atlas, or Atlas unit's power stage becomes short circuited.

An overcurrent fault will cause the following events to occur:

- The current loop and power stage modules are disabled, thereby halting further motor output.
- The Drive Fault Status register records an overcurrent fault condition.
- The Event Status register records a drive exception condition.

To recover from this condition the user should determine the nature of the fault using the **GetDriveFaultStatus** command. It may be desirable to power down Atlas to check connections or otherwise correct the Atlas-attached hardware so that the problem does not occur again.

If the condition is resolved but Atlas is not power cycled, the following sequence should be used to restore the drive to normal operation:

- Clear the fault condition using the **ClearDriveFaultStatus** command.
- Clear the drive exception bit of the Event Status word using the **ResetEventStatus** command. It is not possible to re-enable the current loop or the power stage module if the drive exception bit is still active.

- Re-enable the current loop and power stage modules using the **RestoreOperatingMode** command.

If the overcurrent condition has been resolved, at the end of this sequence Atlas will resume normal operations. If the overcurrent condition has not been resolved, the overcurrent condition will immediately occur again, and the recovery sequence described above must be undertaken again.

Over current faults are serious conditions and warrant the utmost precaution before re-enabling amplifier operation. It is the responsibility of the user to determine the cause and corrective action of any electrical fault.

## 4.8.2 Overtemperature Fault

Atlas provides the capability to continually monitor and detect excessive internal temperature conditions. Such a condition may occur if excessive current is requested, if heat sinking of the Atlas unit is inadequate, or if some other problem results in elevated drive temperatures.

To detect this condition a programmable temperature threshold is continuously compared to an internal temperature sensor. If the value read from the internal sensor exceeds the programmed threshold, an overtemperature fault occurs. In addition, a settable overtemperature hysteresis allows the user to ensure that the Atlas temperature drops by a specified number of degrees before allowing drive restart.

To set the programmable temperature threshold or the programmable temperature hysteresis the command **SetDriveFaultParameter** is used. To read back these values the command **GetDriveFaultParameter** is used.

The maximum allowed setting for the temperature threshold is 75.0° C, which is also the default value. The value set using the **SetDriveFaultParameter** command is in units of degrees C/256. For example, a value of 12,800 sets a threshold of 50° C. The maximum allowed value of the hysteresis parameter is 50° C, and the default value is 5° C.

An over temperature fault will cause the following events to occur:

- The current loop and power stage modules are disabled, thereby halting further motor output.
- The Drive Fault Status register records an overtemperature fault condition.
- The Event Status register records a drive exception condition.

To recover from this condition the user should determine the nature of the fault using the **GetDriveFaultStatus** command. It may be desirable to power down Atlas to correct the condition.

If the condition is resolved but Atlas is not power cycled, the following sequence should be used to restore the drive to normal operation:

- Clear the fault condition using the **ClearDriveFaultStatus** command.
- Clear the drive exception bit of the Event Status word using the **ResetEventStatus** command. It is not possible to re-enable the current loop or the power stage module if the drive exception bit is still active.
- Re-enable the current loop and power stage modules using the **RestoreOperatingMode** command.

If the overtemperature condition has been resolved, at the end of this sequence Atlas will resume normal operations. If the overtemperature condition has not been resolved, the condition will immediately occur again, and the sequence described above should be undertaken again.

The instantaneous status of the overtemperature threshold comparison can be read using the command **GetDriveStatus**.

To read the current value of the temperature sensor the command **GetTemperature** is used.

> Overtemperature faults indicate that the internal safe limit of the drive temperature range has been exceeded. This potentially serious condition can result from incorrect motor connections, excessive power demands placed on the Atlas amplifier, or inadequate heat sinking. It is the responsibility of the user to operate Atlas within safe limits.

## 4.8.3 Overvoltage Fault

Atlas provides the capability to continually monitor and detect excessive voltages on the incoming voltage supply. Such a condition may occur if there is a fault in the system power supply, if a large back EMF (electro motive force) is generated during motor deceleration, or if some other problem results in an elevated bus voltage.

To detect this condition a programmable bus voltage threshold is continuously compared to the bus voltage sensor. If the value read from the internal sensor exceeds the programmed threshold, an overvoltage fault occurs.

To read the current value of the bus voltage the command **GetBusVoltage** is used.

To set the programmable overvoltage threshold the command **SetDriveFaultParameter** is used. To read back this value the command **GetDriveFaultParameter** is used.

The maximum allowed setting for the overvoltage threshold is either 52.0 or 60.0 volts depending on the Atlas unit used. The minimum allowed threshold is 10.0 volts. The value set using the **SetDriveFaultParameter** command is in units of 1.361 mV/count. For example, a value of 14,695 sets a threshold of 14,695 * 1.361 mV/count = 20.00 volts. See Section 3.11.1, "Atlas Settings Defaults and Limits" for a complete list of Atlas limits and default temperature settings.

An overvoltage fault will cause the following events to occur:

- The current loop and power stage modules are disabled, thereby halting further motor output.
- The Drive Fault Status register records an overvoltage fault condition.
- The Event Status register records a drive exception condition.

To recover from this condition the user should determine the nature of the fault using the **GetDriveFaultStatus** command. In most cases it is desirable to power down Atlas to correct the condition.

If the condition is resolved but Atlas is not power cycled, the following sequence should be used to restore the drive to normal operation:

- Clear the fault condition using the **ClearDriveFaultStatus** command.
- Clear the drive exception bit of the Event Status word using the **ResetEventStatus** command. It is not possible to re-enable the current loop or the power stage module if the drive exception bit is still active.
- Re-enable the current loop and power stage modules using the **RestoreOperatingMode** command.

If the overvoltage condition has been resolved, at the end of this sequence Atlas will resume normal operations. If the overvoltage condition has not been resolved, the condition will immediately occur again, and the sequence described above should be undertaken again.

The instantaneous status of the overvoltage threshold comparison can be read using the command **GetDriveStatus**.

> Overvoltage faults indicate that a serious safety condition has occurred. It is the responsibility of the user to operate Atlas within safe limits.

## 4.8.4 Undervoltage Fault

Atlas also provides the capability to sense undervoltage conditions. To set the programmable threshold the command **SetDriveFaultParameters** is used. This value is compared to the value read from the drive DC bus, and if the value read is less than the programmed threshold, an undervoltage fault occurs. See Section 3.11.1, “Atlas Settings Defaults and Limits” for a list of undervoltage-related limits and defaults.

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. Just as for overvoltage conditions, it is the user's responsibility to determine the seriousness of, and appropriate response to, an undervoltage condition.

## 4.8.5 Watchdog Timeout

Atlas provides a programmable watchdog timer that can detect an unexpected lack of activity from the external controller. Typically, such a condition is due to an SPI communication problem or an external controller malfunction. Particularly when the external controller is used to provide higher level velocity or position control, a watchdog timeout may therefore represent a very serious condition.

To effect the watchdog function Atlas monitors the amount of time between successive valid SPI torque or voltage commands from the external controller. If the amount of time between commands exceeds the programmed watchdog timer, the watchdog fault is triggered.

To set the watchdog timeout value the **SetDriveFaultParameter** command is used. To read the value set using this command, **GetDriveFaultParameters** is used. The watchdog time value is in units of 8 * 51.2 µSec. For example a value of 100 indicates a timeout interval of 100 * 8 * 51.2 µSec or 41 mSec. A value of zero (0) means the watchdog is disabled.

A watchdog timeout fault will cause the following events to occur:

- The current loop and power stage modules are disabled, thereby halting further motor output.
- The Drive Fault Status register records a watchdog timeout fault condition.
- The Event Status register records a drive exception condition.

To recover from this condition the user should determine the nature of the fault using the **GetDriveFaultStatus** command. In most cases it is desirable to power down Atlas to correct the condition.

If the condition is resolved but Atlas is not power cycled, the following sequence should be used to restore the drive to normal operation:

- Clear the fault condition using the **ClearDriveFaultStatus** command.
- Clear the drive exception bit of the Event Status word using the **ResetEventStatus** command. It is not possible to re-enable the current loop or the power stage module if the drive exception bit is still active.
- Re-enable the current loop and power stage modules using the **RestoreOperatingMode** command.

At the end of this sequence Atlas will resume normal operations. This includes operation of the watchdog timer itself. Unless a zero value has been loaded into the watchdog timeout value (thereby disabling the watchdog timeout), Atlas will immediately begin counting SPI command intervals to determine if a another timeout has occurred.

> Watchdog timeout faults indicate that a serious safety condition has occurred. It is the responsibility of the user to operate Atlas within safe limits.

## 4.8.6 Drive Enable

Atlas supports an *Enable* input signal that must be active for proper amplifier operation. This signal is useful for allowing external hardware to automatically shut Atlas down. The signal has an active low interpretation.

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

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

- The current loop and power stage modules are disabled, thereby halting further motor output.
- The Drive Fault Status register records a Disabled fault condition.
- The Event Status register records a drive exception condition.

To recover from this condition the user should determine the nature of the fault using the **GetDriveFaultStatus** command. It may be desirable to power down Atlas to correct the condition.

If the condition is resolved but Atlas is not power cycled, the following sequence should be used to restore the drive to normal operation:

- Clear the fault condition using the **ClearDriveFaultStatus** command.
- Clear the drive exception bit of the Event Status word using the **ResetEventStatus** command. It is not possible to re-enable the current loop or the power stage module if the drive exception bit is still active.
- Re-enable the current loop and power stage modules using the **RestoreOperatingMode** command.

At the end of this sequence Atlas will resume normal operations. If the *Enable* signal is still inactive while the disable bit of the Event Status register is being cleared, this bit will immediately be set again, and the recovery sequence must be executed again.

## 4.8.7 Drive Fault Status Register

To simplify recovery from drive-related faults Atlas provides a Drive Fault Status register. This register is read using the command **GetDriveFaultStatus**.

The bits in this register use a latch mechanism, meaning they are set by Atlas, and cleared by the user.

The following table indicates the contents of this register:

| Bit | Name | Description |
|---|---|---|
| 0 | Overcurrent | Set 1 to indicate an overcurrent event due to a short circuit, overload in the drive output, or other such condition. |
| 1-2 | Reserved | May contain 0 or 1. |
| 3 | Operating mode mismatch | Set 1 to indicate that Atlas received a torque command when output was not enabled. |
| 4 | Reserved | May contain 0 or 1. |

| Bit | Name | Description |
|---|---|---|
| 5 | Overvoltage | Set 1 to indicate an overvoltage event in the supply bus voltage. |
| 6 | Undervoltage | Set 1 to indicate an undervoltage event in the supply bus voltage. |
| 7 | Disabled | Set 1 to indicate Enable signal was not asserted. |
| 8 | Current foldback | Set 1 to indicate a foldback event occurred. |
| 9 | Overtemperature | Set 1 to indicate an overtemperature event occurred. |
| 10 | SPI Checksum Error | Set 1 to indicate an SPI checksum has occurred. |
| 11 | Watchdog Timeout | Set 1 to indicate that the command watchdog has timed out. |
| 12 | Current foldback | Set 1 to indicate a current foldback event. |
| 13-15 | Reserved | May contain 0 or 1. |

To clear the bits in this register the command **ClearDriveFaultStatus** is used.

### 4.8.8 FaultOut Signal

The Atlas unit's ***FaultOut*** signal is used to indicate an occurrence of one or more drive faults. This signal is active high, meaning it is high when a fault has occurred, and it is low when a fault has not occurred.

The ***FaultOut*** signal is programmable, so that the user may determine what fault states result in the ***FaultOut*** signal becoming active. In particular, any bit condition of the Drive Fault Status register may be used to trigger activation of the ***FaultOut*** signal. This is done using the command **SetFaultOutMask**. The value set using this command can be read back using **GetFaultOutMask**. See Section 4.8.7, "Drive Fault Status Register" for more information on the Drive Fault Status register

The bit mask specified using this command is ANDed with the Drive Fault Status register. If the result is non-zero, the ***FaultOut*** signal is driven active. For example if a watchdog timeout has occurred (bit 11 of the Drive Fault Status register and the mask has been set to a value of 0x800 (hexadecimal notation, equivalent to decimal value of 2,048) ***FaultOut*** will be active.

The default value for the fault out mask is 0x861, indicating that ***FaultOut*** will go active whenever an overcurrent, overvoltage, undervoltage, or watchdog timeout event occurs.

### 4.8.9 Current Foldback

Current foldback, also known as $I^2t$ foldback, is a general purpose tool to protect the drive output stage or the motor 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 a user-settable continuous current limit. When the integrated value reaches a user-settable energy limit, Atlas goes into current foldback. The default response to this event is to cause the current loop and power stage modules to be disabled. However it is also possible to program Atlas to attempt to clamp the maximum current to the continuous current limit value. Note that the Atlas unit's ability to do so depends on a properly functioning current loop.

Atlas will stay in foldback until the integrator returns to zero. This is shown in Figure 4-10.

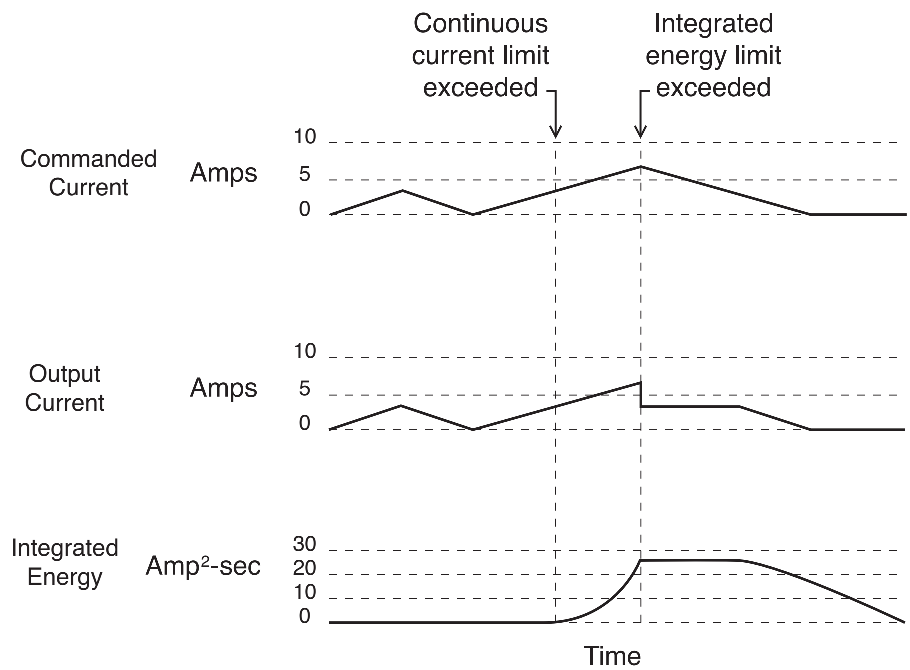



**Figure 4-10: Current Foldback Processing Example**

Each Atlas amplifier has particular default and maximum allowed values for both the continuous current limit and energy limit. These values are designed to protect the Atlas from excessive heat generation. Refer to Section 3.11.1, "Atlas Settings Defaults and Limits" for a complete listing of these defaults and limits.

Setting continuous current limit and energy limit to less than the maximum supported by the Atlas is useful if the current limit is due to the motor, or to some other factor external to Atlas.

To set the continuous current limit and the energy limit the command **SetCurrentFoldback** is used. The values set using this command can be read back using **GetCurrentFoldback**. The provided current value is an unsigned 16-bit integer. Refer to Section 3.11, "Atlas Conversion Factors" for applicable units and scale factors for the specific Atlas unit you are using.

The instantaneous state of the current foldback mechanism (whether the foldback limit is active or not) is available in the Drive Status register and can be read using the command **GetDriveStatus**. If a foldback event has occurred, this event is recorded in the Event Status register as well as the Drive Fault Status register, and can be read back using **GetEventStatus** and **GetDriveFaultStatus** respectively.

#### 4.8.9.1 Current Foldback in Voltage Mode

Atlas unit's current foldback mechanism still operates when Atlas is in voltage mode (current loop disabled). When in this mode, the $I^2t$ energy calculations and condition testing are identical as when Atlas 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, Atlas disables the power stage module, thereby halting further motor output.

#### 4.8.9.2 Example $I^2t$ Calculations

The following example may help illustrate use of current foldback:

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 = $(5A^2 - 3A^2)$ * 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.

### 4.8.9.3 Current Foldback Event Processing

Atlas provides a programmable mechanism related to the current foldback condition. This can be useful for tailoring the response of the Atlas to best suit a specific application.

The Atlas facility that handles this type of programmable response is called an event. Events allow event actions to be executed automatically once a specified condition occurs. In this case, the event condition is satisfied if Atlas is in a current foldback condition, and the following table describes the event actions that can be selected by the external controller:

| Action Name | Description |
|---|---|
| No Action | No event-related action taken, current limited to continuous current limit |
| Disable power stage and current loop | Disables the power stage and the current loop module |

The command **SetEventAction** is used to specify both the condition and the action that should be taken. The external controller must specify that the condition is "current foldback," and also specify one of the two conditions from the table above. Upon occurrence, the programmed action is executed along with any related actions, such as setting the appropriate bit in the Event Status register.

> If a foldback event occurs when Atlas is in voltage mode (current loop not enabled), then the power stage will be disabled regardless of the setting of the event action.

To recover from an event action the command **RestoreOperatingMode** is used. This command will reset Atlas to the operating mode previously specified using **SetOperatingMode** command. Note that if the event condition is still present, the event action will immediately occur again.

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

The default action for the current foldback event is to disable the power stage and current loop.

## 4.9 Step Motor Control

While many aspects of Atlas operation are similar between step motors and Brushless DC or DC Brush motors, Atlas provides a number of special features for supporting step motors. This section describes these special step motor-specific features.

Overall, Atlas provides two step-motor specific position command methods. These are summarized in the table below:

| Position Command Mode | Description |
|---|---|
| Pulse & direction signal input | Atlas directly supports input of hardware *Pulse*, *Direction*, and *AtRest* signals to interface with traditional external controllers that provide these signals. When operated in this mode, SPI communication is not available. |
| SPI step motor | This interface utilizes the external controller SPI interface to provide continuous position information to Atlas. Compared to pulse & direction signal input, this approach allows full use of the SPI communication interface. |

## 4.9.1 Pulse & Direction Signal Input Mode

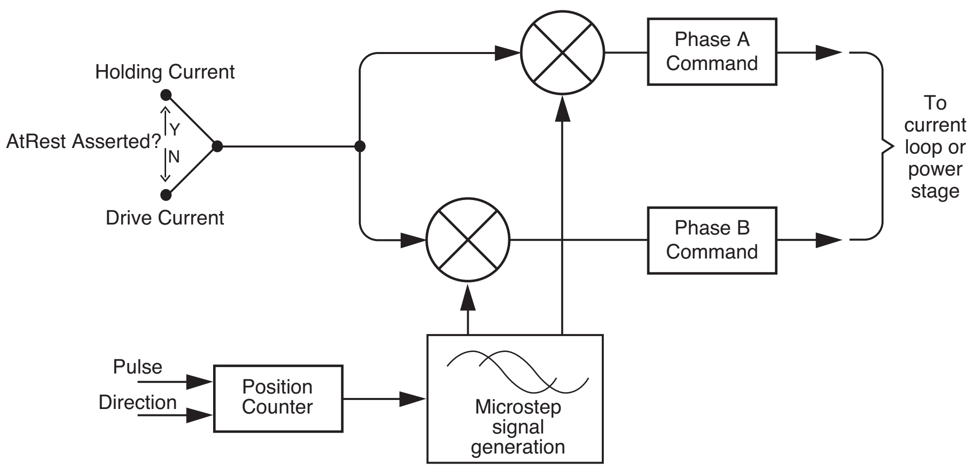



**Figure 4-11: Pulse and Direction Signal Input Mode Control Flow**

Figure 4-11 shows the control flow of the Atlas when used in pulse & direction signal input mode. The Atlas *Pulse* signal drives a counter which increments or decrements a step motor command position based on the state of the *Direction* signal. Pulse signals are expected to be active low, meaning that a position increment or decrement occurs when this signal transitions from high to low. The *Direction* signal indicates that a pulse received while the *Direction* signal is low is interpreted as a negative direction command, and a pulse received while this signal is high as a positive direction command.

Atlas provides programmable microstepping resolution, which means that the incoming position data stream can be interpreted by the Atlas at various resolutions. The maximum is 256 microsteps per full step, and the default interpretation is 64 microsteps per full step. This means that in the default condition, for a standard 1.8° two-phase stepper, Atlas provides a resolution of 12,800 microsteps per mechanical motor rotation, or roughly .028 degrees of mechanical motion per incoming pulse. Note that the control resolution may not equal the actual system accuracy.

To set the microstep resolution the command **SetPhaseCounts** is used. To read this value back, the command **GetPhaseCounts** is used. Phase counts are expressed as microsteps per electrical cycles and there are four full steps per electrical cycle. So for example, to set 256 microsteps per full step, the command **SetPhaseCounts 1,024** is used.

When actually connected to Atlas in pulse & direction signal mode, SPI communication is not available. Command examples in this section are therefore provided for use during setup, before pulse & direction signal mode is operational.

#### 4.9.1.1 Setting the Motor Current

When operating in pulse & direction input mode the current output by Atlas must be specified. Atlas supports two separate, programmable torques. The first is called the drive current and is used during normal step motor operations. The second is called the holding current and is used when the motor is stationary as indicated by the *AtRest* signal. This signal is typically set when the external controller's trajectory generator completes its move or shortly thereafter. *AtRest* is an active low signal, meaning when no motion is occurring this signal should be is set low, and set high when motion is occurring.

To set both the drive current and the holding current the command **SetCurrent** is used. A value between 0 and 32,767 is set. The scaling of this command is determined by the particular Atlas you are using. See Section 3.11, "Atlas Conversion Factors" for details. To read this value, the command **GetCurrent** is used.

#### 4.9.1.2 Pulse & Direction Signal Mode Operation Setup

Because Atlas shares SPI bus signal pins with the *Pulse*, *Direction*, and *AtRest* pins it is not possible to operate the Atlas in the pulse & direction signal input mode while communicating via the SPI communication bus.

To resolve this, Atlas is first connected in SPI mode, and the SPI interface is used to optimize the motion control application during application development and to download desired drive parameters to Atlas unit's non-volatile initialization storage memory. See Section 4.12, "Power-up" for more information on non-volatile initialization storage.

Once Atlas has been fully prepared, to switch Atlas to pulse & direction signal input mode the command **SetDriveCommandMode** is used. This command takes effect immediately, so further SPI command processing is disabled, and the external controller should begin sending pulse & direction commands. Note that this command sequence is generally only performed during power-on initialization. See Section 4.12, "Power-up" for more information.

> Once the command mode has been set to pulse and direction signal, Atlas will be in pulse & direction signal mode and the SPI bus will no longer be functional until a subsequent power cycle occurs.

#### 4.9.1.3 Recovering from Pulse & Direction Signal Mode

It is possible to restore an Atlas that is functioning in pulse & direction signal mode to SPI operation. While this is an uncommon operation, it may be useful for testing, diagnosing a field problem, or to allow a production Atlas to be used for prototyping with optimization software such as PMD's Pro-Motion software. Here is how such a recovery is accomplished.

For approximately 250 ms after power on Atlas will monitor its *Pulse* and *Direction* input signals for a special pattern. If detected, this recovery pattern will cause the Atlas to revert to SPI communication mode.

The recovery pattern consists of a rising edge on the *Pulse* signal with the *Direction* signal set low, five pulses (ten edges) on the *Direction* signal, followed by a falling edge on the *Pulse* signal. The time between direction edges is required to be at least 500 us, and the *AtRest/~SPICS* signal should be low throughout the recovery pattern.

Note that in this recovery pattern *Direction* signal edges occur while the *Pulse* input signal is high, contrary to normal pulse and direction input where a step is signaled by a high to low transition of the *Pulse* input, and all *Direction* edges should occur while *Pulse* is low. Note also that when in pulse and direction signal input mode the *SPISO* pin is in a high impedance state. Once the recovery pattern is recognized *SPISO* is driven low.

#### 4.9.1.4 Fault Processing While in Pulse & Direction Signal Input Mode

In order to allow recovery from safety-related faults such as overtemperature or current foldback while operating in pulse & direction signal input mode, an automatic recovery mode is available. While this mode is most often used when in pulse & direction signal input mode, it may in fact be selected even when SPI communications are available.

Automatic event recovery mode is set using the **SetDriveFaultParameter** command, and may be read back using **GetDriveFaultParameter**.

While in automatic recovery mode the *Enable* signal is used to request that the Atlas automatically attempt to reset a fault condition. After the *FaultOut* signal goes active, the external controller must delay a minimum of 150 µSec, but thereafter may request that the Atlas 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 Atlas it behaves as though the command sequence **ResetEventStatus 0**, **ClearDriveFaultStatus** and **RestoreOperatingMode** has been sent to it by the external controller. As is the case when these commands are sent by the external controller, if the fault condition is still present when recovery is attempted, Atlas 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 Atlas operation.

> While in automatic recovery mode, the FaultOutMask should be set to assert the *FaultOut* signal during a current foldback event. If not, the external controller will be unaware that Atlas is in a foldback condition, and therefore will not know when to request an automatic recovery via the *Enable* signal.

### 4.9.2 SPI Step Motor Command Mode

The SPI step motor command mode allows step motor command information to be transmitted via the SPI bus. When placed in this mode an incremental signed move distance is specified via the SPI command protocol at each external controller command. For example, a value of +7 specified by the external controller means the Atlas will move the step motor position forward 7 microsteps, and a command of -3 will cause the Atlas to move the step motor position backwards 3 microsteps.

To accommodate the *AtRest* signal, the SPI protocol incorporates a bit which is utilized by Atlas in the same manner as the *AtRest* signal when in the hardware pulse & direction mode. See Chapter 5, *SPI Communications* for a complete description of the SPI command format.

As was the case for pulse & direction signal mode, both a drive current and a holding current should be specified when using the SPI step motor command mode. To accomplish this the command **SetCurrent** is used. The values set using this commands can be read by the command **GetCurrent**.

To assist with operation in this mode it may be useful to read the current step motor position. This 32 bit quantity can be read using the command **GetCommandedPosition**.

When using SPI step motor command mode Atlas uses a digital filter to smooth the sequence of output microsteps. Note that this filter introduces a delay of approximately the period between SPI commands.

> When using SPI step motor command mode the SPI commands should be sent at regular intervals. If SPI commands are not sent at equally spaced intervals motor motion may be degraded, and in extreme cases may not output the commanded incremental position.

SPI step motor command mode is the recommended connection mode when Atlas Amplifiers are driven by Magellan Motion Control ICs.

### 4.9.3 Current Control with Step Motors

Current control when driving step motors should use A/B current control. See Section 4.5, "Current Loop" for a detailed summary of the Atlas unit's current loop.

To select the current control mode the command **SetCurrentControlMode** is used. The value set can be read back using **GetCurrentControlMode**.

## 4.10 User Memory Space & Buffers

**Figure 4-12: User Memory Space and Buffers**

| Start Address (in Hexadecimal) | Function |
|---|---|
| 0X0000 0000 | Trace RAM (1,020 words) |
| 0X0000 03FD | Reserved |
| 0X2000 0000 | NVRAM (1,024 words) |
| 0X2000 0400 | Reserved |

Atlas provides the ability to store or retrieve data from an internal user memory space. Figure 4-12 shows the user memory space for Atlas. There is a trace area, and an area that is used to store non-volatile setup commands that can be read back by the external controller. See Section 4.11, "Trace Capture" for more information on trace. See Section 4.12, "Power-up" for more information on Atlas operational configuration storage.

Atlas provides access to the user memory space via a mechanism known as a buffer. Atlas allows up to four different buffers to be defined, allowing the overall user memory space to be partitioned in a manner most useful to the external controller. Typically, there will just be two buffers, one for the trace area, and one for the setup area. However this additional flexibility may be useful for storing temporary application specific data, or to set up multiple trace data storage areas.

Buffers describe a contiguous block of memory and are defined by specifying a base address for the memory block and a block length. Once a buffer's base address and length have been defined, data values may be written to and read from the buffer.

When defining memory buffers the memory space is treated as a sequence of 16-bit memory locations. Atlas allows any values to be used for the base address and length as long as these values result in addresses within the available user memory space.

### 4.10.1 Buffer Indexes

In addition to the base address and length each buffer maintains a read index and a write index. The read index may be assigned a value between 0 and L-1 where L is the buffer length. The read index defines the location from which the next value will be read.

Similarly, the write index ranges from 0 to L-1 and defines the location at which the next value will be written. When a value is read from the memory buffer, the read index is automatically incremented, thus selecting the next value for reading. The write index is incremented whenever a value is written to a buffer. If either index reaches the end of the buffer, it is automatically reset to 0 on the next read/write operation.

## 4.10.2 Buffer Access Commands

The following table details the commands that set up, access, and monitor memory buffers.

| Command | Arguments | Description |
|---|---|---|
| SetBufferStart | bufferID, address | Sets the base address of a buffer. bufferID is either 0, 1, 2, or 3. Address is a 32-bit integer that defines the base address of the buffer. |
| GetBufferStart | bufferID | Returns the base address of the specified buffer. |
| SetBufferLength | bufferID, length | Sets the length of the specified buffer. Length is a 32-bit integer. Atlas adds length to the current buffer base address (as set by the SetBufferStart instruction) to ensure that the buffer will not extend beyond the addressable memory limit. |
| 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-1, where length is the current buffer length. |
| GetBufferWriteIndex | bufferID | Returns the value of the write index for the specified buffer. |
| ReadBuffer16 | bufferID | Returns a 16-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). |

# 4.11 Trace Capture

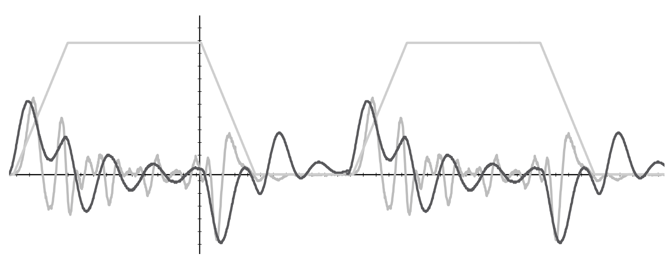


**Figure 4-13: Example Motion Trace Capture**

Trace capture is a powerful Atlas feature that allows parameters and registers to be continuously captured and stored to the Atlas units' trace RAM user memory space. The captured data may later be downloaded by the external controller using standard memory buffer access commands. Data traces are useful for optimizing current loop performance, diagnosing SPI communications, capturing signal data, or assisting with any type of monitoring where a precise time-based record is required.

Broadly speaking, there are three phases associated with data trace operations. In the first phase, the external controller specifies which parameters will be captured, and how the trace will be executed. In the second phase, Atlas captures the trace data. This can occur autonomously, or under external controller control. Finally, in the third phase the external controller retrieves the data. This can occur after the trace is complete, or concurrently with capture.

## 4.11.1 Trace Parameters

To start a trace the external controller must specify a number of parameters. They are:

| Parameter | Description |
|---|---|
| Trace buffer | The external controller must initialize and specify the memory buffer that will be used for the trace data storage area. See Section 4.10, "User Memory Space & Buffers" for more information on buffers. |
| Trace period | Atlas can capture the value of the trace variables for every single time cycle, every other cycle, or at any programmed frequency. This trace period of data collection and storage must be specified. |
| Trace variables | There are dozens of separate variables and registers within Atlas that may be traced; for example, the phase A current command, the current loop error, etc… The user must specify the variables that will be traced by Atlas. |
| Trace mode | Atlas 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 be stopped explicitly by the external controller. |
| Trigger mode | Atlas supports two different methods for determining the moment when data capture actually occurs. The first is internally, via the Atlas unit's internal clock cycle and the trace period defined above. The second is externally commanded by the external controller via the SPI communication protocol. See Chapter 5, *SPI Communications* for more information on the SPI protocol. Note that when external trigger mode is selected the trace period is not used. |
| Trace Start/Stop | Atlas allows the external controller to control when trace capture starts and stops. Two overall conditions are supported; starting and stopping immediately via a command or via the trace bit of the SPI communication header. |

### 4.11.1.1 Trace Buffer

Atlas organizes its internal user memory space into data buffers. Each buffer is given a numerical ID. The trace buffer must always be ID 0 (zero). Before trace capture may be used, memory buffer 0 must be programmed with a valid base address and length.

The size of the trace buffer determines the maximum number of data points that can be captured. For the large majority of applications the trace buffer will be set to a size of 1,020 words, which corresponds to the maximum available memory for trace. If the external controller specifies some of this area for other functions however, such as to store a previous trace, the trace buffer must be set to a smaller size.

While trace data is being collected it is not legal to change the trace buffer configuration. If an attempt is made to change the base address, length, write index, or read index associated with buffer 0 while a trace is running the change will be ignored and an error will be flagged.

### 4.11.1.2 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 51.2 µSecs.

The command **SetTracePeriod** sets the trace period, and the command **GetTracePeriod** retrieves it. Note that if the trigger mode is set to external, the trace period is not used.

### 4.11.1.3 Trace Variables

When traces are running one to four Atlas parameters may be stored to the trace buffer for each occurrence of the trigger. The four trace variable registers are used to define which parameters are stored.

The command **SetTraceVariable** selects which traceable parameter will be stored by the trace variable specified. The values passed by this command specify the variable number of the parameter to be traced, and the variable ID. The command **GetTraceVariable** retrieves this same value.

The following table shows all of the Atlas variables that can be traced along with the variable ID code that is used to select this variable for tracing.

| Variable ID | Name | Description |
|---|---|---|
| **Status Registers** | | |
| 12 | Event Status | The Event Status register |
| 14 | Signal Status | The Signal Status register |
| 56 | Drive Status | The Drive Status register |
| 79 | Drive Fault Status | The Drive Fault Status register |
| 80 | SPI Status | The SPI Status word |
| **Commutation/Phasing** | | |
| 7 | Active Motor Command | The external controller-commanded voltage or torque command |
| 17 | Phase A Command | The output command for phase A |
| 18 | Phase B Command | The output command for phase B |
| 19 | Phase C Command | The output command for phase C |
| 29 | Phase Angle Scaled | The phase angle, scaled from 0 to 360° rather than in encoder counts. |
| **Current Loop** | | |
| 66 | Phase A Reference | The current loop reference for Phase A |
| 67 | Phase B Reference | The current loop reference for Phase B |
| 30 | Phase A Error | The current loop error for Phase A |
| 35 | Phase B Error | The current loop error for Phase B |
| 31 | Phase A Actual Current | The current loop actual current for Phase A |
| 36 | Phase B Actual Current | The current loop actual current for Phase B |
| 33 | Phase A Integrator Contribution | The current loop integrator contribution for Phase A |
| 38 | Phase B Integrator Contribution | The current loop integrator contribution for Phase B |
| 34 | Phase A Current Loop Output | The current loop output for Phase A |
| 39 | Phase B Current Loop Output | The current loop output for Phase B |
| **Field Oriented Control** | | |
| 40 | d Reference | The FOC reference for d (direct) loop |
| 46 | q Reference | The FOC reference for q (quadrature) loop |
| 41 | d Error | The FOC d (direct) loop error |
| 47 | q Error | The FOC q (quadrature) loop error |
| 42 | d Feedback | The d (direct) feedback current |
| 48 | q Feedback | The q (quadrature) feedback current |
| 44 | d Integrator Contribution | The FOC integrator contribution for d (direct) |
| 50 | q Integrator Contribution | The FOC integrator contribution for q (quadrature) |
| 45 | d Output | The FOC output for d (direct) |
| 51 | q Output | The FOC output for q (quadrature) |
| 52 | FOC phase A Output | The FOC output for phase A |
| 53 | FOC phase B Output | The FOC output for phase B |
| 73 | Alpha Current | The FOC $\alpha$ current component (stationary frame) |
| 74 | Beta Current | The FOC $\beta$ current component (stationary frame) |
| 31 | Phase A Actual Current | The FOC actual current for phase A |
| 36 | Phase B Actual Current | The FOC actual current for phase B |

| Variable ID | Name | Description |
|---|---|---|
| **Motor Output** | | |
| 54 | Bus voltage | The bus voltage |
| 55 | Temperature | The temperature of Atlas |
| 68 | $I^2t$ Energy | Accumulated $I^2t$ foldback energy |
| 75 | Terminal A Output | The PWM duty cycle for terminal A |
| 76 | Terminal B Output | The PWM duty cycle for terminal B |
| 77 | Terminal C Output | The PWM duty cycle for terminal C |
| 69 | Leg Current A | The measured current in lower leg A |
| 70 | Leg Current B | The measured current in lower leg B |
| 71 | Leg Current C | The measured current in lower leg C |
| 72 | Leg Current D | The measured current in lower leg D |
| 78 | Clip Factor | Actual output as a fraction of commanded output |
| **Miscellaneous** | | |
| 0 | None | No trace variable is selected |
| 8 | Atlas Time | Atlas unit's processor time in units of cycles |

Setting a trace variable's parameter to zero will disable that variable and all subsequent variables. Therefore, if N parameters are to be saved at each trace period, trace variables 0 to (N–1) must be used to identify the parameters to be saved, and trace variable N must be set to zero.

#### 4.11.1.4 Trace Mode

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 trace 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, overwriting data from previous cycles. In this mode the trace will not end until explicitly commanded by the external controller. See Section 4.11.2, "Trace Start/Stop" for more information on trace stop/start control.

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

#### 4.11.1.5 Trigger Mode

Atlas supports two separate methods for synchronizing data capture during trace operations; internal, under Atlas control, and external, under external controller control.

If internal is selected, trace data capture occurs automatically as determined by the trace period variable described previously. If external is selected, a special bit in the SPI protocol format is used to command when the trace will occur.

In external trigger mode an explicit command must be sent to prepare Atlas for tracing. This command resets various trace variables such as the write index so that they start the trace correctly initialized.

To select trigger mode use the command **SetTraceMode**. The command **GetTraceMode** retrieves this same information.

### 4.11.2 Trace Start/Stop

The external controller has the ability to control when trace capture starts and stops. Both the start condition and the stop condition can be independently programmed.

The command that is used to specify how the trace will start is **SetTraceStart**, and the command that controls how it will stop is **SetTraceStop**. Two conditions are specifiable, the first is immediate, meaning that the trace starts or stops immediately upon receipt of the **SetTraceStart** or **SetTraceStop** command. The second condition specifies that the trace bit of the SPI header will control when trace starts and stops. Trace starts at the moment this bit is set to 1, and stops at the moment this bit is set to zero.

Once a specified start or stop condition occurs, the condition is no longer active. This generally means that the condition should be reprogrammed for each trace operation.

The commands **GetTraceStart** and **GetTraceStop** are used to retrieve the currently active trace start/stop conditions.

Whether started immediately or via the SPI header trace bit, when the trace start condition occurs, all indexes are set to zero and trace data storage starts at the beginning of the buffer.

It is always necessary to specify a start condition for the trace to begin, however it is not necessary to specify a stop condition. If in rolling buffer mode, if no stop condition is specified then the trace will continue indefinitely. If in one-time buffer mode, the trace will continue till the end of the buffer is reached.

See Section 5.2, "Packet Header" for detailed information of the SPI header packet format.

## 4.11.3 Trace Status Word

Atlas provides a register that summarizes the instantaneous state of the trace process. This register can be read using the command **GetTraceStatus**, and is summarized below:

| Name | Bit Number | Description |
|---|---|---|
| Wrap Mode | 0 | Set to 0 when trace is in one-time mode, 1 when in rolling mode. |
| Activity | 1 | Set to 1 when trace is active (currently tracing), 0 if trace is stopped either because of a command or filling the trace buffer. |
| Data Wrap | 2 | Set to 1 when trace has filled the buffer; when in rolling mode trace will then write at the beginning of the buffer, and data may be lost. |
| Overrun | 3 | Set to 1 when a trace buffer location has been written before the previously written value was read from buffer 0. If all trace reads are done using buffer 0 this bit indicates that data has been lost. |
| Data Available | 4 | Set to 1 when a trace buffer location has been written but has not been read from buffer 0. If all trace reads are done using buffer 0 this bit indicates that new trace data is available. |
| — | 5-7 | (Reserved) |
| Trigger Mode | 8 | Clear means internal trigger mode, a set of trace samples is taken based on the Atlas internal clock and the Trace Period register. Set means external trigger mode, exactly one set of trace samples is taken whenever a torque command with the trace bit set is received |
| — | 9-15 | (Reserved) |

## 4.11.4 Downloading Trace Data

Captured data may be downloaded by the external controller using standard buffer memory commands. See Section 4.10, "User Memory Space & Buffers" for a complete description of external memory buffer commands.

When operating in one-time trace mode, the most common approach for downloading data is to allow the trace to finish, and then to read the entire buffer starting at read index 0.

When operating in rolling mode, there are some additional considerations introduced by the fact that the write index can wrap, continuing to write data at buffer addresses already written during that trace.

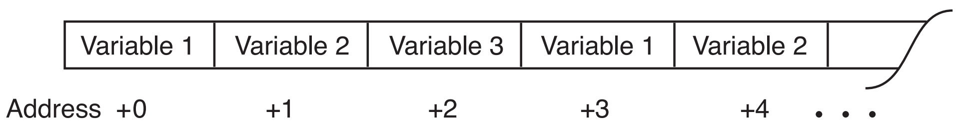
To assist with this the command **GetTraceCount** is available to retrieve the total number of captures that have occurred since the start of the trace. By comparing this number with the number of data capture sets retrieved by the external controller, the external controller can determine how many more data capture sets are available for retrieval.

Also useful during rolling trace mode is the Overrun bit of the Trace Capture register. This bit indicates that Atlas has overwritten data that has not yet been read by the external controller. If the external controller's intention is to continually retrieve all data recorded, this indicates a problem.

### 4.11.5 Trace Data Format

During each trace period each of the trace variables is used in turn to store a 16-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. This is shown in Figure 4-14 with three variables shown captured.

**Figure 4-14: Trace Data Format**




Along those lines, both the length of the trace buffer and the number of trace variables specified for capture affect the number of capture sets that may be stored. For example, if the trace buffer is set to the Atlas unit's maximum value of 1,020 words and two trace variables are specified, up to 510 trace samples can be stored. However if three trace variables are specified then 340 trace sets may be stored.

If smaller trace buffer sizes are used it is recommended that the length be set to an even multiple of the number of trace variables being used. A simple solution is to verify that the trace buffer length is an even multiple of 12, since 12 is evenly divisible by all possible numbers of trace variables: 1, 2, 3, and 4.

## 4.12 Power-up

After receiving stable power at the HV pins Atlas begins its initialization sequence.

In a power-up where no user-provided initialization parameters 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 Atlas is ready to receive commands and begin operation.

Atlas also supports the ability to store initialization parameters that are applied during the power up sequence. For this purpose, Atlas supports a 1,024 word memory that is non-volatile (NVRAM), meaning the data stored will be available even after power to the Atlas is removed. Figure 4-12 shows the user memory space and the location of the NVRAM segment.

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

For more information on how the initialization commands are stored into NVRAM see Section 4.13, "Non-Volatile (NVRAM) Storage."

### 4.12.1 Initialization Command Processing

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

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

The order of initialization for most commands does not matter. However commands that enable Atlas for operation should be executed last in the sequence. This is because Atlas should not begin operations until all of the initialization parameters are loaded in. These commands include **SetOperatingMode**, which is used to enable Atlas modules, and **SetDriveCommandMode**, which is used to set Atlas to pulse & direction signal input mode.

### 4.12.2 Initialization-Specific Commands

To make power-up initialization as flexible as possible there are a few commands that are only available during initialization. These commands are listed below:

The command **Update** is used to activate buffered commands that would otherwise be made active (updated) using an SPI header command.

The command **ExecutionControl** takes an argument which specifies a delay during stored command initialization processing. This command is useful in the case that the user wants Atlas to be ready for operation only when other hardware in the overall system is powered up.

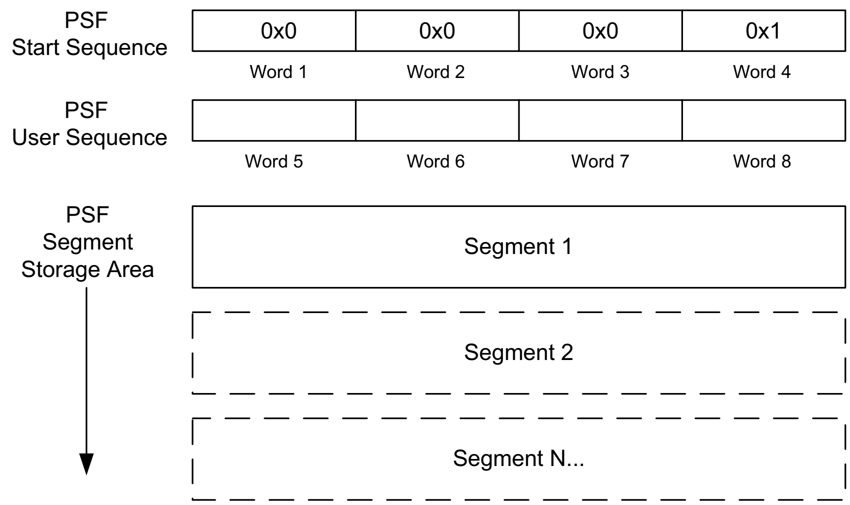
## 4.13 Non-Volatile (NVRAM) Storage

A primary purpose of the NVRAM is to allow Atlas 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 however 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 Atlas NVRAM utlizes a data format known as PMD Structured data Storage Format (PSF). Users who rely only on PMD's Pro-Motion software package to communicate with Atlas and store and retrieve initialization parameters may not need to 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 Atlas NVRAM will utilize the PSF format details provided in the subsequent sections.

## 4.13.1 PMD Structured Data Format

**Figure 4-15: High-Level Format of a PSF (PMD Structured Data Format) Memory Space**




PSF (PMD Structured data Format) is a general purpose data storage format designed for use with non-volatile storage memory such as provided by Atlas Digital Amplifiers. PSF provides a method to store and label initialization information used by the Atlas during startup, as well as to allow user-defined storage in NVRAM.

Figure 4-15 shows the overall format of a PSF-managed memory area. The PSF memory space begins with a 4-word start sequence and a 4-word user programmable sequence. Each word is 16 bits in size, as are future references to words in the following sections unless otherwise noted. The start sequence must contain, in order, the values 0x0, 0x0, 0x0, and 0x1. The user sequence can be specified by the user and may contain any values. The user sequence can be used for any purpose but is often used to identify the type of information stored in the PSF memory space.

Following the eight words of sequence words are one or more data storage blocks known as segments, which are themselves structured memory blocks which must follow a specific format.

## 4.13.2 PSF Data Segments

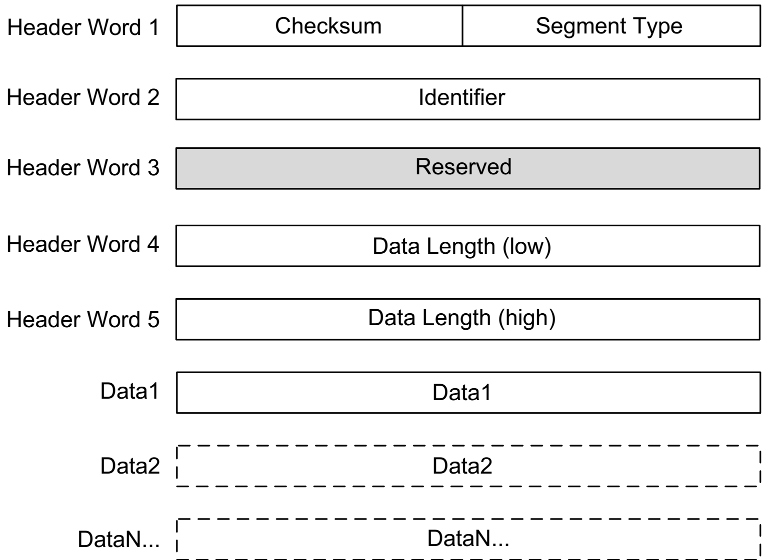



**Figure 4-16: PSF Data Segment Format**

The central mechanism which PSF provides to store data is called a data segment. PSF data segments come with their own headers which allow structuring and data integrity checks of the PSF memory space. Figure 4-16 shows the format of a PSF data segment. The following section details each of the elements in this data structure.

*Checksum* - is the ones complement of an 8-bit ones complement checksum with a seed of 0xAA. It is computed over the entire segment space including the header. If the checksum field is computed correctly then the checksum will be 255 (0xff). The size of this field is one byte.

*Segment type* - specifies the formatting of the data stored in the segment. This 8 bit field encodes the values 0 through 255. Users may assign segment type values 192-255 for segment types of their own design while all other values are reserved. The size of this field is one byte.

*Data length low word & high word* - contains a 32 bit value encoding the number of 16-bit words of data (data0, data1, etc…) included with this segment. Data segments can be defined such that a variable number of data words is expected or a fixed number of words is expected. Whether the number of data words varies or not, the data length word must always be specified correctly for the number of data words actually contained in the segment.

*Identifier* - contains an unformatted 16-bit value that may be used for any purpose but is generally used to identify separate instances of multiply stored segments of the same segment type. For example if there was an array of stored segments, each of the same segment type, the identifier field might be used to identify a specific element within of the overall array of segments.

*Data0, Data1, etc…* is the data that is being stored in this segment. The exact format of this data is determined by the segment type.

## 4.13.3 Pre-Defined Segment Types

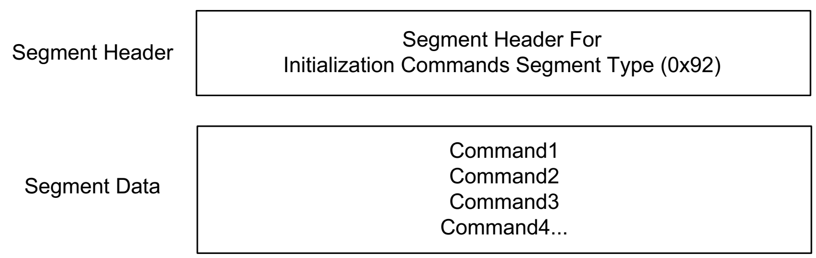
There are two pre-defined Atlas PSF storage segment types. The *Initialization Commands* storage type defines the segment that holds configuration information used during power-up while the *Parameter List* segment holds information that is useful to label the contents of the *Initialization Commands* segment.

During power up Atlas scans the NVRAM space for a properly formatted segment with type '*Initialization Commands*,' and if found it initializes the Atlas using the information provided. The *Initialization Commands* segment type is defined in detail in Section 4.13.4, "Initialization Commands Segment Type."

A segment of type *Parameter List*, when preceding another segment and when containing certain specific values in the data, stores identification information associated with that segment. For example a human-readable name for the segment can be assigned along with information such as when the segment data was stored. This segment-identifying data is not utilized directly by Atlas but rather by software programs such as Pro-Motion. The *Parameter List* segment type is discussed in detail in Section 4.13.5, "Parameter List Segment Type."

## 4.13.4 Initialization Commands Segment Type

**Figure 4-17: Initialization Commands Segment Format**




The *Initialization Commands* segment type selects a segment format that holds the PMD commands that are processed during powerup. The segment type value for the *Initialization Commands* segment type is 0x92. The overall format of this segment type is shown in Figure 4-17.

Atlas commands stored into the segment data portion of the *Initialization Commands* segment is formatted exactly as if it were being sent by the external controller using the SPI protocol during normal SPI operations. See Chapter 5, *SPI Communications* for more information on the exact format of Atlas commands sent over the SPI protocol.

The table below shows a portion of an example initialization command sequence. These example commands enable automatic event recovery mode, delay for 256 cycles so that other system components may initialize themselves, and enable motor output and current control.

| Segment Data Address | Mnemonic | Stored Code (in hex) | Comments |
|---|---|---|---|
| Data1 | SetDriveFaultParameter 2 1 | 0xEF62 | Opcode (0x62) and checksum |
| Data2 | | 0x0002 | Argument 1: event handling mode |
| Data3 | | 0x0001 | Argument 2: automatic event recovery |
| Data4 | ExecutionControl 0 256 | 0x1F35 | Opcode (0x35) and checksum |
| Data5 | | 0x0000 | Argument 1: time delay |
| Data6 | | 0x0000 | Argument 2: delay, high word |
| Data7 | | 0x0100 | Argument 2: low word |
| Data8 | SetOperatingMode 0x7 | 0xE865 | Opcode (0x65) and checksum |
| Data9 | | 0x0007 | Argument 1: Enable output, current loop |

See Section 4.13.4, "Initialization Commands Segment Type" for an example of a complete PSF memory image including an initialization command sequence.

See Section 4.12, "Power-up" for more information on initialization command processing during power up.

## 4.13.5 Parameter List Segment Type

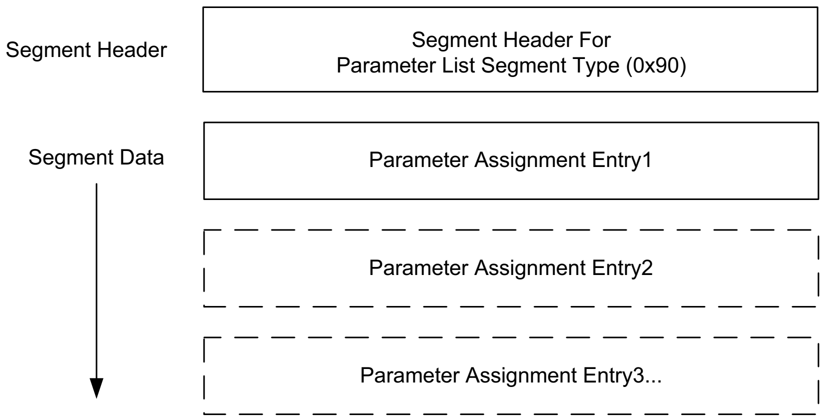



**Figure 4-18: Parameter List Segment Format**

The *Parameter List* segment type provides a general purpose mechanism for the assignment of values to parameters. A major use of the *Parameter List* segment type is to allow human-readable identification information to be recorded and read back, thereby assisting with the identification of PSF-stored data. See Section 4.13.5.2, "Using the ID Segment Mechanism" for information on how this segment ID mechanism is used within the PSF system. The segment type value for the *Parameter List* segment type is 0x90. The overall format of this segment type is shown in Figure 4-18.

The parameter list segment type contains one or more assignments of the general form:

*Parameter = Assigned Value*

*Parameter* specifies the name of the parameter being assigned. *Assigned Value* contains the value to assign to the parameter. *Assigned Value* may be formatted as a character string, an integer, a floating point number, or other formats depending on the *Parameter* being assigned.

The data structure that is used to encode each such assignment in the *Parameter List* segment data area is called a parameter assignment entry. The following section details the format of this data structure.

### 4.13.5.1 Parameter Assignment Entry

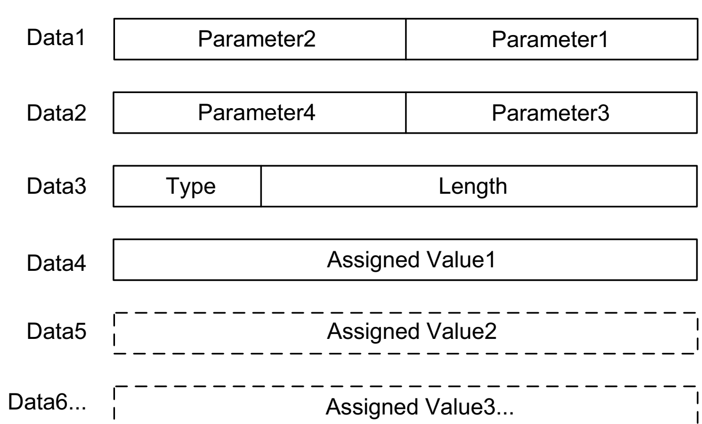



**Figure 4-19: Format of Parameter Assignment Entry**

Figure 4-19 shows the encoding of the data words for a parameter assignment entry.

The *Parameter* field is specified as four byte-length ASCII characters.

The *Type* determines the encoding of the *Assigned Value* data. This field has a length of four bits.

The *Length* field determines the number of words contained in the *Assigned Value*. This field has a length of 12 bits.

*Assigned Value1, Assigned Value2*, etc… hold the data words comprising the *Assigned Value.*

Six specific parameters can be assigned for the purpose of segment identification. Note that not all of these parameters need to be recorded. If not found, Pro-Motion will simply not display the contents for those specific segment ID-related parameters. The following table provides details on the six available segment-ID related parameters

| Parameter Field Encoding | Data Encoding Length & Type | Description |
|---|---|---|
| C, N, [0], [0] | The Assigned Value fields contain a UTF-16 uni-code character string of a variable length set via the length field. The type code for a UTF-16 encoded string is 0. | The CN parameter specifies a general purpose name identifier for the segment to follow. An example name might be "X axis motor init. cmds." Note that the two unused parameter field words after "CN" are filled with zeroes. |
| C,V,E,R | See above | The CVER parameter specifies a version identifier for the segment to follow. An example version might be "version12.3." |
| D,E,S,C | See above | The DESC parameter specifies a general purpose comment for the segment to follow. An example comment might be "These gain factors were determined using the prototype unit in the engineering lab." |
| F,N,[0],[0] | See above | The FN parameter specifies the script file name used to store or retrieve the data in the segment to follow. An example file name might be "xaxis.txt." Note that the two unused parameter field bytes after "FN" are filled with ASCII nuls. |
| F,D,[0],[0] | See above | The FD parameter specifies the modification time of the script file used to store the data in the segment to follow. Times should be recorded in ISO-8601 format "YYYY-MM-DDThh:mm:ss", with hh recorded in 24 hour format. If desired only the year, month and day need be specified. The time portion of this assigned value is optional. An example assigned value might be "2017-01-25T17:13:00" to store a date and time of January 25, 2017 at 5:13pm. Note that the two unused parameter field bytes after "FD" are filled with ASCII nuls. |
| W,D,[0],[0] | See above | The WD parameter specifies the time that data in the segment to follow was written to NVRAM. See "FD" description for encoding and usage example. Note that the two unused parameter field bytes after "WD" are filled with ASCII nuls. |

#### 4.13.5.2 Using the ID Segment Mechanism

Collectively the six parameters from the table above are known as an ID segment. ID Segments specify information for the data segment that immediately follows it in the NVRAM PSF memory space.

When used to provide segment identifying information Pro-Motion, or a similar software program, takes ID information provided by the user and stores it in the correct format into the *Parameter List* segment. The same software program can later search the PSF memory space for segments of type *Parameter List* which hold the correct parameters to retrieve these assigned values for display to the user.

For example if the segment name (see Section 4.13.5.1, "Parameter Assignment Entry" for the various types of ID information that can be stored) was specified and saved to NVRAM as "Axis 1 motor gains" by the user during development, Pro-Motion would read from an Atlas with unknown contents and retrieve this same string for display to the user.

> Other than checking the segment checksum the Atlas unit does not read or otherwise process the ID segment. ID segment information is recorded and retrieved by programs such as Pro-Motion for the convenience and utility of the user. Inclusion of an ID-containing segment is therefore optional.

### 4.13.6 User Defined Segment Types

PSF is a highly flexible data storage system that allows the user to store and if desired, label via the ID segment mechanism structured data into the Atlas NVRAM.

Other than ensuring that the overall NVRAM memory size is not exceeded and that the segment header format is followed there are no restrictions placed on what can be stored in the PSF memory space.

Although not required, PMD recommends that each user-defined segment be preceded with an ID segment that identifies the contents as detailed in Section 4.13.5, “Parameter List Segment Type.” Doing so will assist in keeping track of what data was stored, when, etc… It will also allow the user to develop software tools that can scan the content of the PSF NVRAM space and display a summary of what is stored there, or to utilize Pro-Motion to provide this function.

### 4.13.7 Complete Example PSF Memory Space

Figure 4-20 provides a word-by-word example of an NVRAM image used to store PSF-formatted initialization commands along with associated segment ID content.

**Figure 4-20: Example PSF Memory Space Image**

| Addr | Word | Contents | Comments |
|---|---|---|---|
| 1 | 0x0000 | 0 | PSF Start Sequence |
| 2 | 0x0000 | 0 | |
| 3 | 0x0000 | 0 | |
| 4 | 0x0001 | 1 | |
| 5 | 0x0005 | 5 | PSF User Sequence |
| 6 | 0x0006 | 6 | |
| 7 | 0x0007 | 7 | |
| 8 | 0x0008 | 8 | |
| 9 | 0x2D90 | Chksm, seg. type | Parameter List Segment |
| 10 | 0x0000 | identifier | |
| 11 | 0x0000 | reserved | |
| 12 | 0x002D | length low | |
| 13 | 0x0000 | length high | |
| 14 | 0x4E43 | 'C', 'N' | Assign CN = "Init1" |
| 15 | 0x0000 | nul, nul | |
| 16 | 0x0005 | type, length | |
| 17 | 0x0049 | "I" | |
| 18 | 0x006E | "n" | |
| 19 | 0x0069 | "i" | |
| 20 | 0x0074 | "t" | |
| 21 | 0x0031 | "1" | |
| 22 | 0x5643 | 'C', 'V' | Assign CVER="1.2" |
| 23 | 0x5245 | 'E', 'R' | |
| 24 | 0x0003 | type, length | |
| 25 | 0x0031 | "1" | |
| 26 | 0x002E | "." | |
| 27 | 0x0032 | "2" | |
| 28 | 0x4544 | 'D', 'E' | Assign DESC = "test" |
| 29 | 0x4353 | 'S', 'C' | |
| 30 | 0x0004 | type, length | |
| 31 | 0x0074 | "t" | |
| 32 | 0x0065 | "e" | |
| 33 | 0x0073 | "s" | |
| 34 | 0x0074 | "t" | |
| 35 | 0x4E46 | 'F', 'N' | Assign FN = "file.txt" |
| 36 | 0x0000 | nul, nul | |
| 37 | 0x0008 | type, length | |
| 38 | 0x0066 | "f" | |
| 39 | 0x0069 | "i" | |

| Addr | Word | Contents | Comments |
|---|---|---|---|
| 40 | 0x006c | "l" | |
| 41 | 0x0065 | "e" | |
| 42 | 0x002E | "." | |
| 43 | 0x0074 | "t" | |
| 44 | 0X0078 | "x" | |
| 45 | 0x0074 | "t" | |
| 46 | 0x4457 | 'W', 'D' | Assign WD = "2017-01-25" |
| 47 | 0x0000 | nul, nul | |
| 48 | 0x000A | type, length | |
| 49 | 0x0032 | "2" | |
| 50 | 0x0030 | "0" | |
| 51 | 0x0031 | "1" | |
| 52 | 0x0037 | "7" | |
| 53 | 0x002D | "-" | |
| 54 | 0x0030 | "0" | |
| 55 | 0x0031 | "1" | |
| 56 | 0x002D | "-" | |
| 57 | 0x0032 | "2" | |
| 58 | 0x0035 | "5" | |
| 59 | 0xB992 | chksum, seg. type | Initialization Comments Segment |
| 60 | 0x0000 | identifier | |
| 61 | 0x0000 | reserved | |
| 62 | 0x0009 | length low | |
| 63 | 0x0000 | length high | |
| 64 | 0xEF62 | | SetDriveFault Parameter 2 1 |
| 65 | 0x0002 | | |
| 66 | 0x0001 | | |
| 67 | 0x1F35 | | ExecutionControl 0 256 |
| 68 | 0x0000 | | |
| 69 | 0x0000 | | |
| 70 | 0x0100 | | |
| 71 | 0xE865 | | SetOperatingMode 7 |
| 72 | 0x0007 | | |

## 4.14 Writing and Reading NVRAM Data

The preceding sections described the format of data written to or read from the Atlas unit's NVRAM memory. The following section describes how writing and reading to the NVRAM memory space is accomplished.

### 4.14.1 Writing to NVRAM

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

As detailed in Section 4.13, "Non-Volatile (NVRAM) Storage" if used, the NVRAM memory area should follow the PMD Structured data Format. Failure to do so may result in unexpected behavior of the Atlas unit during power up or during operation. If not used the NVRAM area does not need to be written to or otherwise initialized.

> Data stored into the NVRAM area must follow the PSF format detailed in Section 4.13, "Non-Volatile (NVRAM) Storage." Failure to do so may result in unexpected behavior of the Atlas.

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

1. Send a **DriveNVRAM** command with an argument of *NVRAMMode*. Sending this command places Atlas in a special mode allowing it to store memory into the NVRAM. Before proceeding the external controller should delay 1 second or more.
2. Send a **DriveNVRAM** command with an argument of *EraseNVRAM*. This command will erase the entire NVRAM memory area. Before proceeding the external 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 **DriveNVRAM** with an argument of *Write* is sent, along with the data word to be written. After each word is written Atlas increments an internal pointer so that subsequent data words are automatically stored in the correct location.
4. Check for NVRAM write completion by sending a **NOP** command until a valid checksum is returned. In most cases this will occur right away, however due to the nature of NVRAM writing there may be times when this takes up to a millisecond. Once a valid checksum is returned check the Instruction error bit of the returned SPI Status register. If an error is recorded the entire sequence described above must be repeated from step 2. If no error is recorded continue by repeating steps 3 & 4 until all data is written.
5. Once all data is successfully written the external controller should send a **Reset** command, which will cause Atlas 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.

### 4.14.2 Reading Non-Volatile Memory

If desired, it is possible to directly read the NVRAM memory area using buffer commands. See Section 4.10, "User Memory Space & Buffers" for more information on Atlas buffer processing.

To read the whole NVRAM area the buffer location should be set to 0x20000000 and the length should be set to 1,024. The standard **ReadBuffer16** command can be used.

> It is not possible to write to the NVRAM area using the buffer commands. The procedure outlined in Section 4.14.1, "Writing to NVRAM" must be used to write data to the NVRAM area.

*This page intentionally left blank.*

# 5. SPI Communications



## *In This Chapter*



## 5.1 SPI Communications Overview

Atlas uses an SPI (Serial Peripheral Interface) digital connection to communication with the external controller. This connection is used to setup Atlas parameters, specify voltage or torque output values, monitor Atlas operation, as well as other functions.

SPI is a convenient interface because it is available on many microprocessors, provides relatively high speed communications, and uses only 4 signals; SPIClk (Clock), SPICS (chip select), SPISI (slave in), and SPISO (slave out). Atlas utilizes standard SPI signaling and timing control for the hardware interface and implements a higher level protocol on top of this. See Section 3.6, “AC Characteristics” to learn more about low-level hardware SPI signal timing, voltage levels, etc.

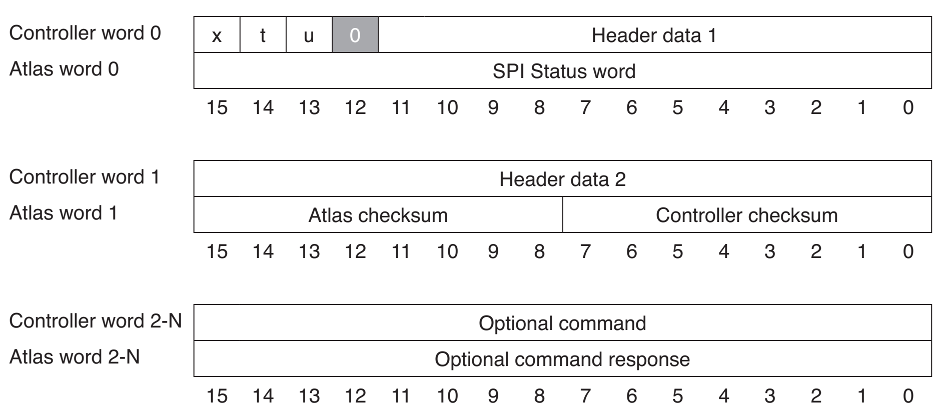



**Figure 5-1: SPI Communications Protocol Overview**

All communications to and from Atlas are in the form of a packet. Figure 5-1 shows the overall packet format. A falling edge of the chip select begins the packet, and a rising edge of the chip select ends the packet. All Atlas SPI packets are comprised of a two word header and one or more optional command words.

# 5.2 Packet Header

The first two words of the packet are called the header and are used to specify a desired motor voltage or torque along with certain other functions such as when a trace starts and when a command update should occur.

Here is a detailed description of the Atlas packet header:

| Field | Bit | Name | Description |
|---|---|---|---|
| data 1 | 0-11 | Header data 1 | Holds various data, the format of which depends on the state of the Torque data flag. |
| | 12 | Reserved | This field is reserved, and should be loaded with a 0. |
| u | 13 | Update flag | A 0 in this field means that a buffered command update is not commanded. A 1 in this field results in an update of all buffered parameters. |
| t | 14 | Trace active flag | This field controls various trace-related activities. See Section 4.11, "Trace Capture" for details. A 0 in this field means that trace is not active. A 1 signals that a trace is active, or that a capture is requested. |
| x | 15 | Torque data flag | A 0 in this field means the header will contain a desired voltage or torque output value. A 1 means that it will contain a request for a disable operation or a NOP operation. |
| data 2 | 0-15 | Header data 2 | Holds various data, the format of which depends on the state of the Torque data flag. |

The 'x' bit field affects the format of other fields, in particular the data 1 and data 2 fields. Therefore care should be taken to correctly select the value of this field and associated data 1 and data 2 fields.

The 't' and the 'u' bit-fields do not affect the format of other fields, and may be set to any value at any time as desired by the external controller. These fields are a means for the external controller to synchronize activities for trace and update-related functions of Atlas. See Section 4.11, "Trace Capture" for more information on use of the trace active flag.

## 5.2.1 Header Return Words

As shown in Figure 5-1 each SPI word sent from the external controller to Atlas results in a return word sent from Atlas to the external controller. In fact at a signal level, each outgoing bit is sent simultaneously with each incoming bit, providing full duplex communications.

The external controller must receive and process data words sent to it by Atlas. These return words, depending on the context, contain transmission integrity information, status bits, or other useful information.

The table below provides the contents of the data words returned by Atlas during header SPI transmissions.

| Field | Description |
|---|---|
| SPI Status word | Contains 16 bits of drive status, signal status, and event information that can be monitored by the external controller. See Section 4.7.4, "SPI Status Register" for a complete description of this word. |
| Atlas checksum | Atlas checksum is the 8 bit, ones-complement checksum of four bytes: the low byte of SPI Status Word, the high byte of SPI Status Word, the Controller Checksum byte (see next field), and the byte value 0xAA. These four bytes along with the Atlas checksum received by the external controller should evaluate to a checksum of 0xFF. |
| Controller checksum | Controller checksum is the 8 bit, ones-complement checksum of five bytes: the low and high bytes of both previously received header words and the byte value 0xAA. These five bytes along with the controller checksum received by the external controller, should evaluate to a checksum of 0xFF. |

> The external controller should verify both the Atlas and Controller checksum. Checksum errors of any kind may indicate a serious problem with external controller to Atlas communications. It it the responsibility of the user to determine the source of any communication problems and take appropriate corrective action

### 5.2.1.1 Example Checksum Calculations

A ones-complement checksum is computed by adding each 8 bit byte as an unsigned quantity, and in case of a carry adding 1. Only the low 8 bits are kept for the final result. A seed of 0xAA is used to make sure that the checksum does not verify in the case that a data line is locked either high or low.

Here is an example of header checksum calculations in the host:

Assume the two words of the previously sent header were 0x6789 and 0xABCD. Assume that the just-received SPI Status Word has a value of 0x147A, and assume that the received Atlas and Controllers checksums are 0xDB and 0xEA respectively.

First, we check the controller checksum-related fields. This is a ones complement addition as follows:

```
    0x67      // high byte of header word 1
    0x89      // low byte of header word 1
    0xAB      // high byte of header word 2
    0xCD      // low byte of header word 2
    0xAA      // checksum seed
+   ------
    0xEA
```

The total indeed equals 0xEA which was the received controller checksum.

Next, we check the Atlas checksum-related fields. This is a ones-complement addition as follows:

```
    0x14      // high byte of received SPI Status Word
    0x7A      // low byte of received SPI Status Word
    0xEA      // received controller checksum
    0xAA      // checksum seed
    0xDB      // received Atlas checksum
+   ----
    0xFF
```

The sum is 0xFF indicating the Atlas checksum value matches the received data.

# 5.3 Sending a Voltage or Torque Output Value

**Figure 5-2: Sending a Voltage or Torque Output Value**

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Controller word 0 | 0 | t | u | 0 | Data 1 | | | | | | | | | | | |
| Atlas word 0 | SPI Status word | | | | | | | | | | | | | | | |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Controller word 1 | Data 2 | | | | | | | | | | | | | | | |
| Atlas word 1 | Atlas checksum | | | | | | | | Controller checksum | | | | | | | |

Generally the most frequently used header transaction is an instantaneous voltage or torque output request for output by the drive. If Atlas is used as part of a higher level velocity or position controller, then these values are continuously sent to Atlas at the servo sampling speed of the main motion controller, typically between 1 kHz and 10 kHz.

Figure 5-2 shows the overall format of the header words when voltage or torque commands are being sent. Depending on the motor type and command mode being used the data words will be loaded one of several different ways. The following table shows this:

| Header Data Word | Contents | Description |
|---|---|---|
| **DC Brush** | | |
| Data 1 | 0 | Must contain zero |
| Data 2 | Torque | Contains a 16-bit signed integer representing the desired torque. The provided current value is an unsigned 16-bit integer. The units depends on the specific type of Atlas amplifier used. For example for higher power units a specified value of 6,553 represents a current command of 1.526mA * 6,553 = 10.00 A. See Section 3.11, "Atlas Conversion Factors" for a list of scale factors |
| **Brushless DC** | | |
| Data1 | Phase angle | Contains a 12-bit unsigned integer representing the electrical phase angle of the motor. A value of 0 equals 0.0°, and a value of 0xFFF represents 359.9°. See Section 4.4, "Commutation" for more information on phase control in brushless motors. |
| Data 2 | Torque | Contains a 16-bit signed integer representing the desired torque. See the above description for the torque scaling. |
| **Step Motor SPI Step Motor Command Mode** | | |
| Data 1 | AtRest Indicator | Bit 0 indicates whether the motor is at rest. If set to 1 motor is at rest and Atlas will use the pre-programmed holding torque. If set to 0 Atlas will use the pre-programmed drive torque. |
| Data 2 | Position increment | Contains a 16-bit signed integer representing the relative position increment to move. See Section 4.9.2, "SPI Step Motor Command Mode" for more information |

# 5.4 Sending an Amplifier Disable

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Controller word 0 | 1 | t | u | 0 | 1 | | | | checksum | | | | | | | |
| Atlas word 0 | SPI Status word | | | | | | | | | | | | | | | |
| Controller word 1 | 0 | | | | | | | | | | | | | | | |
| Atlas word 1 | Atlas checksum | | | | | | | | Controller checksum | | | | | | | |

**Figure 5-3: Amplifier Disable Command Format**

The header can be used to rapidly disable Atlas motor output. This operation is identical to sending a **SetOperationMode** command with current loop and power stage modules disabled, however faster.

To disable Atlas output, a 1 should appear in the 'x' field (bit 15 of the first header word), the high four bits of Data 1 should be loaded with a 1, a checksum should be loaded into the low 8 bits of Data 1, and 0 should be loaded into Data 2. This is shown in Figure 5-3.

The checksum byte should be loaded such that when the ones-complement checksum of all four bytes of the header words, the byte value 0xAA, and the transmitted checksum byte evaluate to 0xFF. If the checksum is not correct, Atlas will take no action as a result of this packet transmission.

# 5.5 Sending a NOP

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Controller word 0 | 1 | t | u | 0 | 0 | | | | *anything* | | | | | | | |
| Atlas word 0 | SPI Status word | | | | | | | | | | | | | | | |
| Controller word 1 | *anything* | | | | | | | | | | | | | | | |
| Atlas word 1 | Atlas checksum | | | | | | | | Controller checksum | | | | | | | |

**Figure 5-4: NOP Command Format**

There are times when it may be useful to send a header without providing a new torque command or disabling the drive. This might be the case, for example, when the external controller only wants to update the 't' (trace) header field or the 'u' (update) fields. In addition, this may be useful to verify controller to Atlas communication without requesting a specific action.

Such an operation is called a NOP (no operation) header transaction.

To send a NOP to Atlas, a 1 should appear in the 'x' field (bit 15 of the first header word), the high four bits of Data 1 should be loaded with a 0, the low byte of Data 1 can contain anything, and Data 2 can contain anything. This is shown in Figure 5-4.

# 5.6 Sending Atlas Commands

Header communication transactions, described in the preceding sections, always consist of two 16-bit words. As such, they are well suited to high speed operations such as continuously sending desired torque commands.

In addition to header communications however, Atlas supports an extensive command set that specifies and queries a much larger number of registers and parameters. Throughout this manual we have illustrated the use of this command set with mnemonics, for example the commands **SetOperatingMode** or **GetEventAction**.

Access to this full Atlas command set is supported within the SPI protocol using the command words shown in Figure 5-5. Commands are split into two overall groups, those that specify information to Atlas, and those that query information from Atlas. Processing of each type is somewhat different, and the following sections will describe both of these types of command operations.

## 5.6.1 Sending an Atlas Command

**Figure 5-5: Send Command Format**

| | 15–8 | 7–0 |
|---|---|---|
| Controller word 0 | checksum | opcode |
| Atlas word 0 | 0 | |

| | 15–0 |
|---|---|
| Controller word 1 | argument word 0 |
| Atlas word 1 | 0 |

| | 15–0 |
|---|---|
| Controller word 2 | argument word 1 |
| Atlas word 2 | 0 |

| | 15–0 |
|---|---|
| Controller word 3 | argument word 2 |
| Atlas word 3 | 0 |

Figure 5-5 shows the overall format for commands that request an action or specify data to Atlas.

The associated data fields begin at the third word of the overall SPI packet and consist of a checksum byte, an opcode byte, and one or more argument data words.

The checksum is calculated by ones-complement adding a byte value of 0xAA and all bytes of all words of the command including the opcode and argument words, but not including any data words in the header. When evaluated, this checksum should give a value of 0xFF for the checksum.

Each separate command mnemonic supported by Atlas is encoded with a unique 8 bit opcode. These opcodes, along with the argument fields of the command data word, are listed in Chapter 6, *Instruction Reference*. Users familiar with PMD's Magellan, Navigator, or 1st generation motion control IC products will note that the overall format of these commands are very similar to those products.

## 5.6.2 Error Processing

If the command checksum shown in Figure 5-6 received by Atlas does not evaluate to 0xFF Atlas will return an SPI Checksum Error and set the instruction error bit of the Event Status register.

In addition to such checksum errors the instruction error bit is set when an otherwise valid instruction or instruction sequence is sent when the Atlas unit's current operating state makes the instructions invalid, when an invalid opcode is sent, or when the arguments to a command are invalid.

## 5.6.3 Sending Commands that Query Information

| | 15 14 13 12 11 10 9 8 | 7 6 5 4 3 2 1 0 |
|---|---|---|
| Controller word 0 | 0 | |
| Atlas word 0 | checksum | result status |
| Controller word 1 | 0 | |
| Atlas word 1 | result word 0 | |
| Controller word 2 | 0 | |
| Atlas word 2 | result word 1 | |
| Controller word 3 | 0 | |
| Atlas word 3 | result word 2 | |

**Figure 5-6: Query Command Format**

Figure 5-6 shows the overall return packet format for an Atlas command sent by the external controller that requests information from Atlas.

Atlas returns a series of result words. The first such word contains a checksum byte such that the ones-complement sum of 0xAA, the checksum itself, and all the data bytes returned by Atlas evaluates to 0xFF. This checksum is correct when the ones-complement sum of these bytes is 0xFF. This first result word also contains a signed 8-bit number containing the number of result words to follow. If this number is negative, an error has occurred, and the instruction error bit of the Event Status register is set.

Subsequent result words hold the actual data being returned by Atlas. For example if the command **GetOperatingMode** is sent, the first result word will contain an 8-bit checksum in the high byte and the number 1 in the low byte indicating 1 data word to follow. The next result word will contain the actual 16-bit value of the operating mode register.

> During a read command, while the external controller is loading data from Atlas, it should write zeroes to the SPI bus. Similarly, during a write command by the external controller Atlas loads zeroes in the return word. Therefore it is recommended that the external processor check that these words have a zero. If not, this indicates that the external controller has become out of synch with Atlas indicating a communication problem.

A typical response message will be available within 50 μSec. If however the query returns an "SPI Command Timing Violation" (0xE9) then the request was sent too soon but can be tried again.

## 5.6.4 Command Interlacing

To lower the worst case timing burden of sending and retrieving results from non-torque-sending commands Atlas expects such commands to be interlaced with header operations. No more than one extra word per SPI packet is allowed with these commands. When required due to the length of the command the overall command data transmissions occurs over the course of several such header transactions.

# 6. Instruction Reference



## 6.1 How to Use This Reference

The instructions are arranged alphabetically, except that all "Set/Get" pairs (for example, **SetFOC** and **GetFOC**) are described together. Each description begins on a new page and most occupy no more than a single page. Each page is organized as follows:

| | |
|---|---|
| **Name** | The instruction mnemonic is shown at the left, its hexadecimal code at the right. |
| **Syntax** | The instruction mnemonic (in bold) and its required arguments (in italic) are shown with all arguments separated by spaces. |
| **Buffered** | Certain parameters and other data written to the motion control IC are buffered. That is, they are not acted upon until the next command update. These parameters are identified by the word "buffered" in the instruction heading. |
| **Motor Types** | The motor types to which this command applies. Supported motor types are printed in black; unsupported motor types for the command are greyed out. |
| **Arguments** | There are two types of arguments: encoded-field and numeric.<br>Encoded-field arguments are packed into a single 16-bit data word. The name of the argument (in italic) is that shown in the generic syntax. Instance (in italic) is the mnemonic used to represent the data value. Encoding is the value assigned to the field for that instance.<br>For numeric arguments, the parameter value, the type (signed or unsigned integer), and the range of acceptable values are given. Numeric arguments may require one or two data words. For 32-bit arguments, the high-order part is transmitted first. |
| **Packet Structure** | This is a graphic representation of the 16-bit words transmitted in the packet: the instruction, which is identified by its name, followed by 1, 2, or 3 data words. Bit numbers are shown directly below each word. For each field in a word, only the high and low bits are shown. For 32-bit numeric data, the high-order bits are numbered from 16 to 31, the low-order bits from 0 to 15.<br>The hex code of the instruction is shown in boldface.<br>Argument names are shown in their respective words or fields.<br>For data words, the direction of transfer—read or write—is shown at the left of the word's diagram.<br>Unused bits are shaded. All unused bits must be 0 in data words and instructions sent (written) to the motion control IC. |
| **Description** | Describes what the instruction does and any special information relating to the instruction. |
| **Restrictions** | Describes the circumstances in which the instruction is not valid, that is, when it should not be issued. For example, GetCommandedPosition is relevant only in pulse and direction mode. |
| **see** | Refers to related instructions. |

# ClearDriveFaultStatus

**Syntax** **ClearDriveFaultStatus**

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments** None

**Packet Structure**

**ClearDriveFaultStatus**

| checksum (bits 15–8) | **6C**h (bits 7–0) |
|---|---|

**Description** **ClearDriveFaultStatus** clears all bits in the Drive Fault Status register. A bit is cleared only if it has been read by **GetDriveFaultStatus** since the last detection of the fault condition, so that information on faults detected between **GetDriveFaultStatus** and **ClearDriveFaultStatus** is not lost.

**Restrictions**

**see** **GetDriveFaultStatus** (p. 103)

# DriveNVRAM 30h

**Syntax** **DriveNVRAM** *Option Value*

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments**

| Name | Instance | Encoding |
|---|---|---|
| *option* | *NVRAM Mode* | 0 |
| | *Erase NVRAM* | 1 |
| | *Write* | 2 |
| | *Block Write Begin* | 3 |
| | *Block Write End* | 4 |
| | *Skip* | 8 |
| | *Unlock* | 11 |

| | Type | Range |
|---|---|---|
| *value* | *unsigned 16-bit* | See below |

None

**Packet Structure**

**DriveNVRAM**

| | 15–8 | 7–0 |
|---|---|---|
| write | checksum | **30**h |

| | 15–0 |
|---|---|
| write | option |
| write | value |

**Description**

The **DriveNVRAM** command is used to program the non-volatile memory. This memory is used primarily for user-specified power-on initialization, but may also be used for storing arbitrary non-volatile user data.

The command **DriveNVRAM** 0 1 may be used to reset Atlas without executing the commands in NVRAM. This command is required in order to read NVRAM contents when they would configure Atlas in pulse and direction input mode. It may be used either in NVRAM or normal modes.

Atlas must be put in a special mode of operation in order to program NVRAM, in this mode most Atlas commands are not supported, and will return an error code of NVRAM Mode (26). In order to enter NVRAM mode, use the command **DriveNVRAM** 0 0.

While changing to NVRAM mode Atlas will not respond to SPI communications, so the controlling processor should use this sequence of operations:

1) Send the **DriveNVRAM** command

2) Wait for at least 500 microseconds

3) Send a NOP torque command

4) While the checksum read is wrong, repeat step (3)

This sequence of operations should also be used with the **Erase NVRAM**, **Write**, and **Block Write End** commands, all of which may interfere with SPI communications for some time. The Instruction Error bit of the SPI status word should be checked after each such NVRAM operation, and the **GetInstructionError** command used to check error status if it is set.

**Description (cont.)**

NVRAM bits may be cleared individually, but may only be set as an entire block. This operation is called an erase, in the erased state each word reads as 0xFFFF. Typically NVRAM will be erased each time new initialization instructions are written, but this is not absolutely required.

In order to erase NVRAM, use the command **DriveNVRAM 1 0**, and follow the wait sequence shown for entering NVRAM mode.

After erasing NVRAM, 16-bit words may be written, beginning at location zero. After writing each word an internal pointer will be advanced to the next location. It is the user's responsibility to keep track of the current write position, it cannot be directly read.

In order to write a single word, use the command **DriveNVRAM 2 Value**, where Value may be any 16-bit word. After writing the wait sequence should be followed.

If it is desired to skip a sequence of locations that already have the correct values the command **DriveNVRAM 8 N** may be used, where N is the number of words to skip. No wait procedure is required after this command. Related to this the unlock option allows writing to an NVRAM sector that has not been erased. The command **DriveNVRAM 11 0** provides this unlock function.

In order to speed up long NVRAM writes a block write facility is provided. A block write is begun by using the **DriveNVRAM 3 N** command, where N is the number of words that will be written as a block. N may be at most 32. No wait procedure is required after this command.

The values to write are provided using a torque-like command, the first header word of this command is 0x0F00, and the second word is the value to write to an internal buffer.

Once all the values of a write block have been sent the command **DriveNVRAM 4** checksum should be sent to start the actual write process. The checksum argument is a 16-bit ones complement checksum over all of the words to be written, if Atlas does not verify the checksum it will respond with an NVRAM checksum Error (25). If the checksum is verified then the success code will be returned, and the wait procedure should be followed.

When all NVRAM operations are complete it is necessary to either power cycle Atlas or send a Reset command in order to exit NVRAM mode.

It is not necessary to enter NVRAM mode in order to read the contents of NVRAM, that may be done by using the SetBufferStart command to specify the NVRAM base, 0x20000000, as the start of a buffer, and the ReadBuffer16 command to retrieve NVRAM values.

**Restrictions**

Before entering NVRAM mode motor output must be disabled by setting the operating mode to 1. Most ordinary commands are not supported in NVRAM mode.

**see**

**SetOperatingMode** (p. 144), **GetInstructionError** (p. 109), **SetBufferStart**, **ReadBuffer16** (p. 121)

**Syntax** **ExecutionControl** *option cycles*

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments**

| Name | Instance | Encoding |
|---|---|---|
| *option* | *0* | *time delay* |

| Name | Type | Range | Scaling |
|---|---|---|---|
| *cycles* | unsigned 32 bits | $0\text{-}2^{32}-1$ | unity |

**Returned data** None

**Packet Structure**

| | 15 … 8 | 7 … 0 |
|---|---|---|
| write | checksum | **35**h |

| | 15 … 0 |
|---|---|
| write | option |
| write | Delay (high order part) |
| write | Delay (low order part) |

**Description**

**ExecutionControl** is used to cause a fixed time delay during the processing of user initialization commands in the Atlas non-volatile memory, typically to allow some other system component to initialize or stabilize itself.

The argument specifies the number of current loop cycles to delay. Normal Atlas processing of external signals, status registers, and motor outputs is enabled during the user initialization phase.

*This command was previously named InitializationDelay.*

**Restrictions**

**ExecutionControl** is valid only as an initialization command read from nonvolatile memory, it is not a valid SPI command.

# GetActiveOperatingMode 57h

**Syntax** **GetActiveOperatingMode**

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments** None

**Returned Data**

| | Type | |
|---|---|---|
| *mode* | unsigned 16 bits | bit field |

**Packet Structure**

**GetActiveOperatingMode**

| checksum (15–8) | **57**h (7–0) |
|---|---|

First data word

| | 15–0 |
|---|---|
| read | *mode* |

**Description**

**GetActiveOperatingMode** gets the actual operating mode that the Atlas is currently using. This may or may not be the same as the static operating mode, as safety responses or programmable conditions may change the **Active Operating Mode**. When this occurs, the **Active Operating Mode** can be changed to the programmed static operating mode using the **RestoreOperatingMode** command. The bit definitions of the operating mode are given below.

| Name | Bit | Description |
|---|---|---|
| — | 0 | Reserved. |
| Motor Output Enabled | 1 | 0: motor outputs disabled. 1: motor outputs enabled. |
| Current Control Enabled | 2 | 0: current control bypassed. 1: current control active. |
| — | 3–15 | Reserved |

When the current loop is disabled, it operates by passing its input directly to its output, and clearing all internal state variables (such as integrator sums, etc.).

**Restrictions**

**see** **GetOperatingMode** (p. 144), **RestoreOperatingMode** (p. 125), **Set/GetEventAction** (p. 139)

**Syntax** GetBusVoltage

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments** None

**Returned Data**

| | Type | Range | Scaling |
|---|---|---|---|
| *voltage* | unsigned 16 bits | 0 *to* $2^{16}$–1 | 1.3612 mv/count |

**Packet Structure**

**GetBusVoltage**

| checksum (bits 15–8) | **40**h (bits 7–0) |
|---|---|

First data word

| | bits 15–0 |
|---|---|
| read | *voltage* |

**Description** **GetBusVoltage** gets the most recent bus voltage reading from the Atlas.

**Restrictions**

**see** **Get/SetDriveFault** (p. 103)

# GetChecksum F8h

**Syntax** GetChecksum

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments** None

**Returned data**

| Name | Type |
|---|---|
| *checksum* | unsigned 32 bits |

**Packet Structure**

GetChecksum

| checksum (bits 15–8) | **F8**h (bits 7–0) |
|---|---|

First data word

| | bits 31–16 |
|---|---|
| read | *checksum* (high-order part) |

Second data word

| | bits 15–0 |
|---|---|
| read | *checksum* (low-order part) |

**Description** **GetChecksum** reads the Atlas internal 32-bit *checksum* value. The return value is dependent on the silicon revision number of the motion control IC.

**Restrictions**

# GetCommandedPosition

**Syntax** **GetCommandedPosition**

**Motor Types**

| | Microstepping |
|---|---|

**Arguments** None

**Returned data**

| | Type | Range | Scaling | Units |
|---|---|---|---|---|
| *position* | signed 32 bits | $-2^{31}$ *to* $2^{31}-1$ | unity | microsteps |

**Packet Structure**

**GetCommandedPosition**

| checksum (bits 15–8) | **1D**h (bits 7–0) |
|---|---|

First data word

| | bits 31–16 |
|---|---|
| read | *position* (high-order part) |

Second data word

| | bits 15–0 |
|---|---|
| read | *position* (low-order part) |

**Description** **GetCommandedPosition** returns the commanded position. Commanded position is the instantaneous position value resulting from accumulated pulse and direction commands.

**Restrictions** The result is not meaningful unless Atlas is in pulse and direction mode, either hardware or SPI-emulated

**see** **GetDriveCommandMode** (p. 135)

# GetCurrentLoopValue 71h

**Syntax** **GetCurrentLoopValue** *loopnum node*

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments**

| Name | Instance | Encoding |
|---|---|---|
| *phase* | *Phase A* | 0 |
| | *Phase B* | 1 |
| *node* | *Reference* | 0 |
| | *Actual Current* | 1 |
| | *Error* | 2 |
| | — (Reserved) | 4 |
| | *Integral Contribution* | 5 |
| | *Output* | 6 |
| | $I^2t$ Energy | 10 |

**Returned data**

| | Type | Range/Scaling |
|---|---|---|
| *value* | signed 32 bits | see below |

**Packet Structure**

**GetCurrentLoopValue**

| checksum (bits 15–8) | **71**h (bits 7–0) |
|---|---|

First data word

| | 0 (bits 15–12) | *phase* (bits 11–8) | *node* (bits 7–0) |
|---|---|---|---|
| write | 0 | *phase* | *node* |

Second data word

| | |
|---|---|
| read | *value* (high-order part) (bits 31–16) |

Third data word

| | |
|---|---|
| read | *value* (low-order part) (bits 15–0) |

**Description**

**GetCurrentLoopValue** is used to read the value of a ***node*** in one of the digital current loops. See the product user guide for more information on the location of each ***node*** in the current loop processing. Though the data returned is signed 32 bits regardless of the ***node***, the range and format vary depending on the ***node***, as follows:

| Node | Range | Scaling | Units |
|---|---|---|---|
| *Reference* | –2^15 to 2^15–1 | Dependent on unit* | Amps |
| *Actual Current* | –2^15 to 2^15–1 | Dependent on unit* | Amps |
| *Error* | –2^15 to 2^15–1 | Dependent on unit* | Amps |
| *Integral Contribution* | –2^31 to 2^31–1 | 100/2^22 | % max output |
| *Output* | –2^15 to 2^15–1 | 100/2^15 | % max output |
| $I^2t$ *Energy* | –2^31 to 2^31–1 | 100/2^31 | % max energy |

* *See* Section 3.11, "Atlas Conversion Factors" *for correct scaling for the Atlas unit you are using.*

**Restrictions** This command is only supported when the current control mode is Phase A /B.

**see** **Set/GetCurrentLoop** (p. 133), **Set/GetCurrentControlMode** (p. 130), **Set/Get Current Foldback** (p. 131)

**Syntax** GetDriveFaultStatus

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments** None

**Returned Data**

| | Type | |
|---|---|---|
| *status* | unsigned 16 bits | see below |

**Packet Structure**

GetDriveFaultStatus

| checksum (bits 15–8) | 6Dh (bits 7–0) |
|---|---|

First Data Word

| read | DriveFaultStatus (bits 15–0) |
|---|---|

**Description**

**GetDriveFaultStatus** gets the Drive Fault Status register, which is used to report the cause of several disabling events.

The table below shows the bit definitions of the Drive Fault Status register.

| Name | Bit |
|---|---|
| Overcurrent Fault | 0 |
| — (Reserved) | 1-2 |
| Operating Mode Mismatch | 3 |
| — (Reserved) | 4 |
| Overvoltage Fault | 5 |
| Undervoltage Fault | 6 |
| Disabled by ~Enable | 7 |
| Current Foldback | 8 |
| Overtemperature | 9 |
| SPI Checksum Error | 10 |
| Watchdog Timeout | 11 |
| — (Reserved) | 12–15 |

All of the fault bits are associated with the Drive Exception event, and indicate the cause of that event. When the Drive Exception event is raised motor output is always disabled.

Operating Mode Mismatch means that a torque command was received when Atlas motor output was not enabled.

Watchdog Timeout means that no valid communication was received from a controlling SPI processor for the watchdog period, which is set using the **SetDriveFaultParameter** command.

Disabled means that an inactive ~Enable signal was seen.

Overcurrent means that an excessive bus or bridge leg current was detected.

Over Voltage and Under Voltage mean that the bus voltage was outside acceptable limits, which may be modified by using the **SetDriveFaultParameter** command.

Current Foldback means that a current foldback event has ocurred.

Overtemperature means that the internal temperature reading has exceeded the limit, which may be specified using the **SetDriveFaultFaultParameter** command.

**Description (cont.)**

SPI Checksum Error means that an error was detected in the header (first two words) of an SPI packet, used for sending torque commands. Checksums are only used when sending an amplifier disable command. See Section 5.4, "Sending an Amplifier Disable" for more information. Checksum errors detected in other command processing are reported using the Instruction Error bit in the Event Status register, and in the Instruction Error register.

All bits in the Drive Fault Status register are latched, and may be cleared by using the **ClearDriveFaultStatus** command, which unconditionally clears all bits. The Drive Fault Status register should be cleared before attempting to handle any disabling condition, so the cause of subsequent failures may be determined.

**Restrictions**

**see**

**ClearDriveFaultStatus** (p. 94), **Set/GetDriveFault Parameter** (p. 136), **GetEventStatus** (p. 106), **GetInstructionError** (p. 109)

# GetDriveStatus 0Eh

**Syntax** **GetDriveStatus**

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments** None

**Returned data**

| | Type | |
|---|---|---|
| *status* | unsigned 16 bits | see below |

**Packet Structure**

**GetDriveStatus**

| checksum (15–8) | 0Eh (7–0) |
|---|---|

First Data Word

| | 15–0 |
|---|---|
| read | DriveStatus |

**Description**

**GetDriveStatus** reads the Drive Status register. All of the bits in this status word are set and cleared by Atlas. They are not settable or clearable by command. The bits represent states or conditions in the Atlas that are of a transient nature.

| Name | Bit(s) | Description |
|---|---|---|
| — | 0 | Reserved; not used; may be 0 or 1. |
| In Foldback | 1 | Set to 1 when the unit is in the current foldback state–the output current is limited by the foldback limit. |
| Overtemperature | 2 | Set to 1 when the overtemperature condition is present. |
| — | 3 | Reserved; not used; may be 0 or 1. |
| In Holding | 4 | Set to 1 when the unit is in the holding current state–the output current is limited by the holding current limit. |
| Overvoltage | 5 | Set to 1 when the overvoltage condition is present. |
| Undervoltage | 6 | Set to 1 when the undervoltage condition is present. |
| Disabled | 7 | The *~Enable* signal is not currently asserted. |
| — | 8–11 | Reserved; not used; may be 0 or 1. |
| Clipping | 12 | Set to 1 when drive output is constrained by PWM limit or by operating limits, 0 when drive output is set by the current loop. |
| — (Reserved) | 13–14 | |
| Initialization | 15 | Set to 1 when processing initialization commands in NVRAM, 0 when initialization is complete. |

**Restrictions**

The In Holding bit is meaningful only when in pulse and direction mode, either hardware or SPI-emulated.

**see**

**ClearDriveFaultStatus** (p. 94)

# GetEventStatus 31h

**Syntax** **GetEventStatus**

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments** None

**Returned data**

| | *Type* | |
|---|---|---|
| *status* | unsigned 16 bits | see below |

**Packet Structure**

**GetEventStatus**

| checksum (bits 15–8) | **31**h (bits 7–0) |
|---|---|

Data

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| read | Event Status | | | | | | | | | | | | | | | |

**Description**

**GetEventStatus** reads the Event Status register. All of the bits in this status word are set by the internal events and cleared by command. To clear these bits, use the **ResetEventStatus** command. The following table shows the encoding of the data returned by this command.

| Name | Bit(s) | Description |
|---|---|---|
| — | 0–6 | (Reserved) |
| Instruction Error | 7 | Set to 1 when an instruction error has occurred. |
| | 8 | (Reserved) |
| Overtemperature Fault | 9 | Set to 1 when an overtemperature condition has occurred. |
| Drive Exception | 10 | Set to 1 when a disabling drive fault has occurred. |
| — | 11 | (Reserved) |
| Current Foldback | 12 | Set to 1 when current foldback has occurred. |
| — | 13-15 | (Reserved) |

**Restrictions**

**see** **GetDriveFaultStatus** (p. 103), **GetDriveStatus** (p. 105), **ResetEventStatus** (p. 124)

# GetFOCValue 5Ah



**Syntax** **GetFOCValue** *loop node*

**Motor Types**

| | Brushless DC | Microstepping |
|---|---|---|

**Arguments**

| Name | Instance | Encoding |
|---|---|---|
| *loop* | *Direct (d)* | 0 |
| | *Quadrature (q)* | 1 |
| *node* | *Reference (d,q)* | 0 |
| | *Feedback (d,q)* | 1 |
| | *Error (d,q)* | 2 |
| | — (Reserved) | 3-4 |
| | *Integral Contribution (d,q)* | 5 |
| | *Output (d,q)* | 6 |
| | *FOC Output (Alpha,Beta)* | 7 |
| | *Actual Current (A,B)* | 8 |
| | $I^2t$ Energy | 10 |

**Returned data**

| | Type | Range/Scaling |
|---|---|---|
| *value* | signed 32 bits | see below |

**Packet Structure**

**GetFOCValue**

| checksum (15–8) | **5A**h (7–0) |
|---|---|

First data word

| | 0 (15–12) | *loop* (11–8) | *node* (7–0) |
|---|---|---|---|
| write | 0 | *loop* | *node* |

Second data word

| | |
|---|---|
| read | *value* (high-order part) (31–16) |

Third data word

| | |
|---|---|
| read | *value* (low-order part) (15–0) |

**Description**

**GetFOCValue** is used to read the value of a *node* of the FOC current control.

Though the data returned is signed 32 bits regardless of the *node*, the range and format vary depending on the *node*, as follows:

| Node | Range | Scaling | Units |
|---|---|---|---|
| *Reference (d,q)* | –2^15 to 2^15–1 | 100/2^14 | % max current |
| *Feedback (d,q)* | –2^18 to 2^18–1 | 100/2^14 | % max current |
| *Error (d,q)* | –2^15 to 2^15–1 | 100/2^14 | % max current |
| *Integral Contribution (d,q)* | –2^31 to 2^31–1 | 100/2^14 | % max current |
| *Output (d,q)* | –2^15 to 2^15–1 | 100/2^14 | % PWM |
| *FOC Output (Alpha,Beta)* | –2^15 to 2^15–1 | 100/2^14 | % PWM |
| *Actual Current (A,B)* | –2^15 to 2^15–1 | 100/2^14 | % max current |
| *$I^2t$ Energy* | –2^31 to 2^31–1 | 100/2^30 | % max energy |

**Description (cont.)**

Most of the ***nodes*** have units of % maximum current, and most have a scaling of 100/2^14. That is, a value of 2^14 corresponds to 100% maximum current. The range is extended to allow for overshoot in excess of maximum peak current, and thus values can be more than 100% of the maximum output current.

The maximum current is the largest current that can be represented rather than the maximum that can be sourced or sensed. The maximum current can be calculated via the formula:

*Max = Current Scaling * 0x8000*

For example for the high power Altas, using the scale factor from Section 3.11, "Atlas Conversion Factors", the maximum current = 1.526mA/count * 0x8000 counts = 50A.

**Restrictions**

This command is only supported when the current control mode is set to FOC.

**see**

**Set/GetFOC** (p. 141), **Set/GetCurrentControlMode** (p. 130)
**Set/Get Current Foldback** (p. 131)

# GetInstructionError

**Syntax** **GetInstructionError**

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments** None

**Returned data**

| | Type | Range |
|---|---|---|
| *First Error* | unsigned 8 bits | 0 *to* 1Ch |
| *Second Error* | unsigned 8 bits | 0 to 1Ch |

**Packet Structure**

**GetInstructionError**

| 15 – 8 | 7 – 0 |
|---|---|
| checksum | **A5**h |

Data

| | 15 – 8 | 7 – 0 |
|---|---|---|
| read | Second Error | First Error |

**Description**

**GetInstructionError** returns two 8 bit codes indicating command failures, and then resets both error fields to zero. Generally, this command is issued only after the instruction error bit in the Event Status register indicates that there was an instruction error.

If only one command failure has occurred then the second error field will be zero; if no command failures have occurred then both fields will be zero. After powering up or receiving a reset command Atlas will set the instruction error bit, and set the first error field in the instruction error register to Processor Reset (1). Any error occurring during initialization from NVRAM will be recorded by the second error field.

The error codes are encoded as defined below:

| Error Code | Encoding |
|---|---|
| No error | 0 |
| Processor reset | 1 |
| Invalid instruction | 2 |
| — (Reserved) | 3 |
| Invalid parameter | 4 |
| Trace running | 5 |
| Bad flash file | 6 |
| Block out of bounds | 7 |
| — (Reserved) | 8 |
| Incorrect NVRAM checksum | 9 |
| — (Reserved) | 0Ah-0Fh |
| Invalid Operating Mode restore after event-triggered change | 10h |
| Invalid Operating Mode for command | 11h |
| Invalid register state | 12h |
| — (Reserved) | 13h, 14h |
| Bad SPI command checksum | 15h |
| Incorrect SPI command protocol | 16h |
| SPI command timing violation | 17h |

**Description (cont.)**

| Error Code | Encoding |
|---|---|
| Invalid torque command | 18h |
| NVRAM write checksum error | 19h |
| Command not valid in NVRAM mode | 1Ah |
| Attempt to write read-only buffer | 1Bh |
| Command valid only when executed from internal memory | 1Ch |

**Restrictions**

**see** **GetEventStatus** (p. 106), **ResetEventStatus** (p. 124)

**Syntax** GetPhaseAngle

**Motor Types**

| | Brushless DC | Microstepping |
|---|---|---|

**Arguments** None

| | Type | Range | Scaling | Units |
|---|---|---|---|---|
| *angle* | unsigned 16 bits | 0 *to* $2^{15}$–1 | unity | revolutions<br>microsteps |

**Packet Structure**

GetPhaseAngle

| checksum (15–8) | 2Ch (7–0) |
|---|---|

Data

| | angle (15–0) |
|---|---|
| read | *angle* |

**Description** GetPhaseAngle returns the instantaneous commutation or microstepping angle as set by the last torque command. 0x10000, which is just out of range, corresponds to 360 degrees, or one revolution, and is equivalent to zero.

**Restrictions** None

**see**

# GetPhaseCommand

**Syntax** **GetPhaseCommand** *phase*

**Motor Types**

| | | |
|---|---|---|
| | **Brushless DC** | **Microstepping** |

**Arguments**

| Name | Instance | Encoding |
|---|---|---|
| *phase* | *Phase A* | 0 |
| | *Phase B* | 1 |
| | *Phase C* | 2 |

**Returned data**

| | Type | Range | Scaling | Units |
|---|---|---|---|---|
| *command* | signed 16 bits | $-2^{15}$ *to* $2^{15}-1$ | $100/2^{15}$ | % output |

**Packet Structure**

**GetPhaseCommand**

| checksum (bits 15–8) | **EA**h (bits 7–0) |
|---|---|

First data word

| write | 0 (bits 15–2) | *phase* (bits 1–0) |
|---|---|---|

Second data word

| read | *command* (bits 15–0) |
|---|---|

**Description**

**GetPhaseCommand** returns the value of the commutated phase command for phase A, B, or C. These are the phase command values directly output to the current loop or motor after commutation.

**Scaling example:** If a value of –4,489 is retrieved (EE77h) for a given phase, then this corresponds to –4,489*100/32,767 = –13.7% of full-scale output.

**Restrictions**

*Phase C* is only valid when the motor type has been set for a 3-phase commutation.

This command has no meaning when current control mode is set to FOC whether or not the current loops are enabled.

When the current control mode is set to *Phase A /B* current loops, the values are the inputs to the current loops. When current loops are disabled, the value is the motor output command.

**see**

**SetCurrentControlMode** (p. 130)

**Syntax** GetSignalStatus

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments** None

**Returned data**

| | Type |
|---|---|
| see below | unsigned 16 bits |

**Packet Structure**

GetSignalStatus

| checksum (15–8) | **A4h** (7–0) |
|---|---|

Data

| | 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
|---|---|
| read | Signal Status |

**Description**

**GetSignalStatus** returns the contents of the Signal Status register. Each bit in the Signal Status register is set if the corresponding signal is high, and clear if the signal is low. The ~Enable signal is an input, and the *FaultOut* signal is an output. See **SetFaultOutMask** (p. 140) for more on how the *FaultOut* signal is controlled.

The Signal Status register contains the value of the various hardware signals connected to the Atlas drive.

The bit definitions are as follows:

| Description | Bit Number |
|---|---|
| — (Reserved) | 0–12 |
| /Enable In | 13 |
| FaultOut | 14 |
| — (Reserved) | 15 |

**Restrictions**

**see** **GetDriveFaultStatus** (p. 103), **GetEventStatus** (p. 106)

# GetTemperature 53h

**Syntax** **GetTemperature**

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments** None

**Returned Data**

| | Type | Range | Scaling | Units |
|---|---|---|---|---|
| *temperature* | signed 16 bits | –2^15 *to* 2^15–1 | $2^8$ | °C |

**Packet Structure**

**GetTemperature**

| checksum (bits 15–8) | **53**h (bits 7–0) |
|---|---|

First data word

| | temperature (bits 15–0) |
|---|---|
| read | *temperature* |

**Description** **GetTemperature** gets the most recent temperature reading from the Atlas internal temperature sensor.

**Restrictions**

**see** **Get/SetDriveFault** (p. 99)

**Syntax** GetTime

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments** None

**Returned data**

| Name | Type | Range | Scaling | Units |
|---|---|---|---|---|
| *time* | unsigned 32 bits | 0 *to* $2^{32}$–1 | unity | cycles |

**Packet Structure**

**GetTime**

| | checksum (bits 15–8) | **3E**h (bits 7–0) |
|---|---|---|

First data word

| | bits 31–16 |
|---|---|
| read | *time* (high-order part) |

Second data word

| | bits 15–0 |
|---|---|
| read | *time* (low-order part) |

**Description** **GetTime** returns the number of cycles which have occurred since the motion control IC was last reset. The time units are current loop cycles.

**Restrictions**

**see**

# GetTraceCount BBh

**Syntax** GetTraceCount

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments** None

**Returned data**

| Name | Type | Range | Scaling | Units |
|---|---|---|---|---|
| *count* | unsigned 32 bits | 0 *to* $2^{32}$–1 | unity | samples |

**Packet Structure**

**GetTraceCount**

| checksum (bits 15–8) | BBh (bits 7–0) |
|---|---|

First data word

| | bits 31–16 |
|---|---|
| read | *count* (high-order part) |

Second data word

| | bits 15–0 |
|---|---|
| read | *count* (low-order part) |

**Description** **GetTraceCount** returns the number of points (variable values) stored in the trace buffer since the beginning of the trace.

**Restrictions** If the trace mode is set to "rolling" and the buffer wraps, **GetTraceCount** returns the total number of samples written since trace was started.

**see** **ReadBuffer16** (p. 121), **Set/GetBufferLength** (p. 126), **Set/GetTraceMode** (p. 148), **Set/Get Trace Start** (p. 150), **Get/Set Trace Stop** (p. 151)

**Syntax** GetTraceStatus

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments** None

**Returned data**

| Name | Type |
|---|---|
| see below | unsigned 16 bits |

**Packet Structure**

GetTraceStatus

| | checksum | | BAh | |
|---|---|---|---|---|
| | 15 | 8 | 7 | 0 |

Data

| read | Trace Status | |
|---|---|---|
| | 15 | 0 |

**Description**

**GetTraceStatus** returns the trace status. The definitions of the individual status bits are as follows:

| Name | Bit Number | Description |
|---|---|---|
| Wrap Mode | 0 | Set to 0 when trace is in one-time mode, 1 when in rolling mode. |
| Activity | 1 | Set to 1 when trace is active (currently tracing), 0 if trace is stopped either because of a command or filling the trace buffer. |
| Data Wrap | 2 | Set to 1 when trace has filled the buffer; when in rolling mode trace will then write at the beginning of the buffer, and data may be lost. |
| Overrun | 3 | Set to 1 when a trace buffer location has been written before the previously written value was read from buffer 0. If all trace reads are done using buffer 0 this bit indicates that data has been lost. |
| Data Available | 4 | Set to 1 when a trace buffer location has been written but has not been read from buffer 0. If all trace reads are done using buffer 0 this bit indicates that new trace data is available. |
| — | 5-7 | (Reserved) |
| Trigger Mode | 8 | Clear means internal trigger mode, a set of trace samples is taken based on the Atlas internal clock and the Trace Period register. Set means external trigger mode, exactly one set of trace samples is taken whenever a torque command with the trace bit set is received |
| — | 9-15 | (Reserved) |

The Data Wrap and Overrun bits are cleared when trace goes from being off (Activity 0) to on (Activity 1), because of the detection of the programmed trace start condition.

**Restrictions**

**see** **Set/GetTraceMode** (p. 148), **Set/Get Trace Start** (p. 150), **Set/Get Trace Stop** (p. 151)

**Syntax** **GetTraceValue** *VariableID*

**Arguments**

| Name | Instance | Encoding |
|---|---|---|
| *VariableID* | *see SetTraceVariable* (p. 152) | |

**Returned data**

| Name | Type |
|---|---|
| *Value* | signed *and* unsigned 32bits |

**Packet Structure**

**GetTraceValue**

| | 15–8 | 7–0 |
|---|---|---|
| write | checksum | **28**h |
| write (Data) | 0h | VariableID |
| read | Value high word | |
| read | Value low word | |

**Description** **GetTraceValue** is used to read a single traceable value, without having to set up the trace buffer, trace mode, and so forth. The meaning of the *variableID* argument is exactly the same as in **SetTraceVariable**. Signed values are sign extended to 32 bits, unsigned values are zero extended.

# GetVersion 8Fh



**Syntax** **GetVersion**

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments** None

**Returned data**

| Name | Type |
|---|---|
| *version* | unsigned 32 bits |

**Packet Structure**

**GetVersion**

| checksum | **8F**h |
|---|---|
| 15 … 8 | 7 … 0 |

First data word

| | product family | motor type | number of axes | special | # chips |
|---|---|---|---|---|---|
| read | | | | | |
| | 31 … 28 | 27 … 24 | 23 … 20 | 19 … 18 | 17 … 16 |

Second data word

| | customization code | product version | major version | minor version |
|---|---|---|---|---|
| read | | | | |
| | 15 … 8 | 7 … 6 | 5 … 4 | 3 … 0 |

**Description** **GetVersion** returns product information encoded as shown in the preceding packet structure diagram. Individual data fields are encoded as defined in the following table.

| Name | Description | Encoding |
|---|---|---|
| product family | Atlas | Ah |
| motor type | DC Brush | 1 |
| motor type | Step Motor | 4 |
| motor type | Brushless DC | 3 |
| motor type | Mult. Motor | 8 |
| number of axes | Maximum number of supported axes | 1 to 15 |
| special | (Reserved) | 0 to 3 |
| # chips | | 1 |
| customization code | None<br>Other | 0<br>1 to 255 |
| product version | | 0 to 3 |
| major s/w version | | 0 to 3 |
| minor s/w version | | 0 to 15 |

**Restrictions**

**see**

# 6 NoOperation 00h

**Syntax** **NoOperation**

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments** None

**Returned data** None

**Packet Structure**

NoOperation

| checksum (15–8) | 00h (7–0) |
|---|---|

**Description** The **NoOperation** command has no effect on the motion control IC. It may be useful for verifying or synchronizing communications.

**Restrictions**

**see**

**Syntax** **ReadBuffer16** *bufferID*

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments**

| Name | Type | Range |
|---|---|---|
| *bufferID* | unsigned 16 bits | 0 *to* 3 |

**Returned data**

| | Type | Range |
|---|---|---|
| *data* | signed 16 bits | $-2^{15}$ *to* $2^{15}-1$ |

**Packet Structure**

**ReadBuffer**

| checksum (bits 15–8) | **CDh** (bits 7–0) |
|---|---|

First data word

| | 0 (bits 15–5) | *bufferID* (bits 4–0) |
|---|---|---|
| write | 0 | *bufferID* |

data word

| | bits 15–0 |
|---|---|
| read | *data* (high-order part) |

**Description** **ReadBuffer16** returns the 16-bit contents of the location pointed to by the read buffer index in the specified buffer. After the contents have been read, the read index is incremented by 1. If the result is equal to the buffer length (set by **SetBufferLength**), the index is reset to zero (0).

**Restrictions**

**see** **Set/GetBufferReadIndex** (p. 127), **Set/GetBufferStart** (p. 128), **Set/GetBufferLength** (p. 126)

# Reset

**39h**

**Syntax** **Reset**

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments** None

**Returned data** None

**Packet Structure**

Reset

| checksum | 39h |
|---|---|
| 15 … 8 | 7 … 0 |

**Description**

**Reset** restores the motion control IC to its initial condition, setting all motion control IC variables to their default values. Most variables are motor-type independent; however several default values depend upon the motor type of the Atlas. The motor-type independent values are listed here.

| | Default Value | Buffered |
|---|---|---|
| **Commutation** | | |
| Phase Angle | 0 | No |
| Phase Counts | 256 | No |
| **Current Control** | | |
| Current Control Mode | 1 | Yes |
| Current Loop Kp (both A and B loops) | 0 | Yes |
| Current Loop Ki (both A and B loops) | 0 | Yes |
| Current Loop Integration Limit (both A and B loops) | 4000h | Yes |
| FOC Kp (both d and q loops) | 0 | Yes |
| FOC Ki (both d and q loops) | 0 | Yes |
| FOC Integration Limit (both d and q loops) | 4000h | Yes |
| Holding Current | 0 | No |
| Drive Current | 0 | No |
| **Motor Output** | | |
| Operating Mode | 1 | No |
| Active Operating Mode | 1 | No |
| PWM Frequency | 4875 | No |
| **RAM Buffers** | | |
| Buffer Start (buffer 0) | 0 | No |
| Buffer Length (buffer 0) | 1020 | No |
| Buffer Start (buffer 1) | 20000000h | No |
| Buffer Length (buffer 1) | 1024 | No |
| Buffer Read Index (all) | 0 | No |
| Buffer Write Index (all) | 0 | No |
| **Safety** | | |
| Current Foldback Event Action | 7 | No |
| Watchdog Limit | 0 | No |
| **Trace** | | |
| Trace Mode | 0 | No |
| Trace Period | 1 | No |
| Trace Variables (all) | 0 | No |

## Description (cont.)

| | Default Value | | | |
|---|---|---|---|---|
| | Low Power | Medium Power | High Power | Buffered |
| Overvoltage Limit | 38,207 (52 V) | 38,207 (52 V) | 44,085 (60 V) | N |
| Undervoltage Limit | 7348 (10 V) | 7348 (10 V) | 7348 (10 V) | N |
| Overtemperature Limit | 19,200 (75°C) | 19,200 (75°C) | 19,200 (75 °C)) | N |
| Temperature Hysteresis | 1280 (5°C) | 1280 (5°C) | 1280 (5°C) | N |
| Continuous Current Limit | | | | |
| BLDC | 9172 (2.12 A) | 9264 (7.07 A) | 9264 (14.14 A) | N |
| DC Brush | 6486 (1.5 A) | 9172 (7.00 A) | 9172 (14.00 A) | N |
| Step | 9172 (2.12 A) | 8338 (6.37 A) | 8338 (12.73 A) | N |
| Energy Limit | | | | |
| BLDC | 502 (3.0 $A^2s$) | 498 (31.9 $A^2s$) | 498 (127.5 $A^2s$) | N |
| DC Brush | 617 (3.6 $A^2s$) | 503 (32.2 $A^2s$) | 503 (128.7 $A^2s$) | N |
| Step | 502 (3.0 $A^2s$) | 542 (34.7 $A^2s$) | 542 (138.9 $A^2s$) | N |

## Notes

## Restrictions

The typical time before the device is ready for communication after a reset is 250ms for Atlas products.

Not all of the listed variables are available on all products. See the product user guide.

## see

# ResetEventStatus 34h

**Syntax** **ResetEventStatus** *mask*

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments**

| Name | Instance | Encoding |
|---|---|---|
| *mask* | *Instruction Error* | FF7Fh |
| | *Overtemperature Fault* | FDFFh |
| | *Drive Exception* | FBFFh |
| | *Current Foldback* | EFFFh |

**Returned data** None

**Packet Structure**

**ResetEventStatus**

| checksum (bits 15–8) | **34**h (bits 7–0) |
|---|---|

Data

| | mask (bits 15–0) |
|---|---|
| write | *mask* |

**Description**

**ResetEventStatus** clears (sets to 0) each bit in the Event Status register that has a value of 0 in the *mask* sent with this command. All other Event Status register bits (bits that have a mask value of 1) are unaffected.

Events that cause changes in operating mode or trajectory require, in general, that the corresponding bit in Event Status be cleared prior to returning to operation. That is, prior to restoring the operating mode in cases where the event caused a change in it.

**Restrictions** Not all bits in **ResetEventStatus** are supported in some products. See the product user manual.

**see** **GetEventStatus** (p. 106)

**Syntax** RestoreOperatingMode

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments** None

**Packet Structure**

RestoreOperatingMode

| checksum | 2Eh |
|---|---|
| 15 … 8 | 7 … 0 |

**Description**

**RestoreOperatingMode** is used to command Atlas to return to its static operating mode. It should be used when the active operating mode has changed due to actions taken from safety events or other programmed events. Calling **RestoreOperatingMode** will re-enable all loops that were disabled as a result of events.

**Restrictions**

Before using **RestoreOperatingMode** to return to the static operating mode, the event status bits should all be cleared. If a bit in event status that caused a change in operating mode is not cleared, this command will return an error. If using Magellan, the **RestoreOperatingMode** command should be sent to a Magellan axis, rather than to an Atlas axis. Otherwise Magellan will immediately disable Atlas again.

**see**

**GetActiveOperatingMode** (p. 98), **Set/GetOperatingMode** (p. 144), **Set/GetEventAction** (p. 139)

# SetBufferLength C2h
# GetBufferLength C3h

**Syntax**

**SetBufferLength** *bufferID length*
**GetBufferLength** *bufferID*

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments**

| Name | Type | Range |
|---|---|---|
| *bufferID* | unsigned 16 bits | 0 *to* 3 |
| *length* | unsigned 32 bits | 1 *to* $2^{30} - 1$ |

**Packet Structure**

**SetBufferLength**

| | checksum (bits 15–8) | **C2**h (bits 7–0) |
|---|---|---|

First data word

| write | 0 (bits 15–2) | *bufferID* (bits 1–0) |
|---|---|---|

Second data word

| write | *length* (high-order part) (bits 31–16) |
|---|---|

Third data word

| write | *length* (low-order part) (bits 15–0) |
|---|---|

**GetBufferLength**

| | checksum (bits 15–8) | **C3**h (bits 7–0) |
|---|---|---|

First data word

| write | 0 (bits 15–2) | *bufferID* (bits 1–0) |
|---|---|---|

Second data word

| read | *length* (high-order part) (bits 31–16) |
|---|---|

Third data word

| read | *length* (low-order part) (bits 15–0) |
|---|---|

**Description**

**SetBufferLength** sets the *length*, in numbers of 16-bit elements, of the buffer in the memory block identified by *bufferID*.

**Note**: The **SetBufferLength** command resets the buffers read and write indexes to 0.

The **GetBufferLength** command returns the *length* of the specified buffer.

**Restrictions**

The buffer length plus the buffer start address cannot exceed the memory size of the product. See Section 2.1, "Operational Specifications" for more information. The memory indicated depends on the buffer start. It may be necessary to decrease buffer length before changing buffer start.

**see**

**Set/GetBufferReadIndex** (p. 127), **Set/GetBufferStart** (p. 128)

**Syntax**

**SetBufferReadIndex** *bufferID index*
**GetBufferReadIndex** *bufferID*

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments**

| Name | Type | Range | Scaling | Units |
|---|---|---|---|---|
| *bufferID* | unsigned 16 bits | 0 *to* 3 | unity | - |
| *index* | unsigned 32 bits | 0 *to* buffer length - 1 | unity | double words |

**Packet Structure**

**SetBufferReadIndex**

| | 15–8 | 7–0 |
|---|---|---|
| | checksum | **C6**h |

First data word

| | 15–2 | 1–0 |
|---|---|---|
| write | 0 | *bufferID* |

Second data word

| | 31–16 |
|---|---|
| write | *index* (high-order part) |

Third data word

| | 15–0 |
|---|---|
| write | *index* (low-order part) |

**GetBufferReadIndex**

| | 15–8 | 7–0 |
|---|---|---|
| | checksum | **C7**h |

First data word

| | 15–2 | 1–0 |
|---|---|---|
| write | 0 | *bufferID* |

Second data word

| | 31–16 |
|---|---|
| read | *index* (high-order part) |

Third data word

| | 15–0 |
|---|---|
| read | *index* (low-order part) |

**Description**

**SetBufferReadIndex** sets the address of the read *index* for the specified **bufferID**.

**GetBufferReadIndex** returns the current read *index* for the specified **bufferID**.

**Restrictions**

If the read index is set to an address beyond the length of the buffer, the command will not be executed and will return error code 7, buffer bound exceeded.

**see**

**Set/GetBufferLength** (p. 126), **Set/GetBufferStart** (p. 128)

# SetBufferStart C0h
# GetBufferStart C1h

**Syntax**

**SetBufferStart** *bufferID address*
**GetBufferStart** *bufferID*

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments**

| Name | Type | Range | Units |
|---|---|---|---|
| *bufferID* | unsigned 16 bits | 0 *to* 3 | - |
| *address* | unsigned 32 bits | 0 *to* $2^{31} - 1$ | double words |

**Packet Structure**

**SetBufferStart**

| | Bits 15–8 | Bits 7–0 |
|---|---|---|
| | checksum | **C0**h |

| | Word | Content | Bits |
|---|---|---|---|
| write | First data word | 0 (bits 15–2), *bufferID* (bits 1–0) | 15–0 |
| write | Second data word | *address* (high-order part) | 31–16 |
| write | Third data word | *address* (low-order part) | 15–0 |

**GetBufferStart**

| | Bits 15–8 | Bits 7–0 |
|---|---|---|
| | checksum | **C1**h |

| | Word | Content | Bits |
|---|---|---|---|
| write | First data word | 0 (bits 15–2), *bufferID* (bits 1–0) | 15–0 |
| read | Second data word | *address* (high-order part) | 31–16 |
| read | Third data word | *address* (low-order part) | 15–0 |

**Description**

**SetBufferStart** sets the starting *address* for the specified buffer, in words, of the buffer in the memory block identified by *bufferID.*

The starting address 0x2000 0000 is used to access NVRAM. 0x2200 0000 is used to access manufacturer NVRAM. When setting these addresses the buffer length must first be set to the length of the indicated NVRAM block or less.

**Note:** The **SetBufferStart** command resets the buffers read and write indexes to 0.

The **GetBufferStart** command returns the starting *address* for the specified *bufferID*.

**Restrictions**

The buffer start address plus the buffer length cannot exceed the memory size of the product. See Section 2.1, “Operational Specifications” for more information.

**see**

**Set/GetBufferLength** (p. 126), **Set/GetBufferReadIndex** (p. 127)

# SetCurrent 5Eh
# GetCurrent 5Fh

## Syntax

**SetCurrent** *option value*
**GetCurrent** *option*

## Motor Types

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

## Arguments

| Name | Instance | Encoding |
|---|---|---|
| *option* | *Holding Current* | 0 |
| | Reserved | 1 |
| | *Drive Current* | 2 |

| | Type | Range | Scaling | Units |
|---|---|---|---|---|
| *value* | 16-bit unsigned | 0 *to* $2^{15}$ | $100/2^{15}$ | % max output |

## Packet Structure

**SetCurrent**

| | 15–8 | 7–0 |
|---|---|---|
| write | checksum | **5E**h |

| | 15–2 | 1–0 |
|---|---|---|
| write | 0 | *option* |

| | 31–16 |
|---|---|
| write | *value* |

**GetCurrent**

| | 15–8 | 7–0 |
|---|---|---|
| write | checksum | **5F**h |

| | 15–2 | 1–0 |
|---|---|---|
| write | 0 | *option* |

| | 31–16 |
|---|---|
| read | value |

## Description

The **SetCurrent** command is used to set the output level (current, or voltage if the current loop is off) used during pulse and direction motor operation. The Holding Current option is used to set the output level when the *AtRest* signal is asserted, and the Drive Current option to set the output level when the *AtRest* signal is not asserted.

The **GetCurrent** command is used to retrieve the output levels set by **SetCurrent**.

## Restrictions

Holding Current should be less than or equal to Drive Current; if it is greater then the Drive Current value will be used at all times. These commands are not meaningful except in pulse and direction mode.

# SetCurrentControlMode buffered 43h
# GetCurrentControlMode 44h



**Syntax**

**SetCurrentControlMode** *axis mode*
**GetCurrentControlMode**

**Syntax**

**Motor Types**

| (n/a) | Brushless DC | Microstepping |
|---|---|---|

**Arguments**

| Name | Instance | Encoding |
|---|---|---|
| *mode* | *A/B* | 0 |
| | *FOC* | 1 |
| | *Third leg floating* | 2 |

**Packet Structure**

**SetCurrentControlMode**

| checksum (bits 15–8) | **43**h (bits 7–0) |
|---|---|

First data word

| | 0 (bits 15–2) | *mode* (bits 1–0) |
|---|---|---|
| write | *0* | *mode* |

**GetCurrentControlMode**

| checksum (bits 15–8) | **44**h (bits 7–0) |
|---|---|

First data word

| | 0 (bits 15–2) | *mode* (bits 1–0) |
|---|---|---|
| read | 0 | *mode* |

**Description**

**SetCurrentControlMode** configures Atlas to use either the *A/B* method, the *FOC* method, or the *Third Leg Floating* method for current control.

**GetCurrentControlMode** gets the buffered current loop *mode*.

**Restrictions**

**SetCurrentControlMode** is a buffered command. It will not take effect until an update is done on the current loop by sorting the **Update** in a torque command. The value read by **GetCurrentControlMode** is the buffered setting.

The Third Leg Floating mode is valid only for 3 phase Brushless DC motor control.

**see**

**GetFOCValue** (p. 107), **Get/SetFOC** (p. 141), **GetCurrentLoopValue** (p. 102), **Get/SetCurrentLoop** (p. 133)

# SetCurrentFoldback 41h
# GetCurrentFoldback 42h



**Syntax**

**SetCurrentFoldback** *parameter value*
**GetCurrentFoldback** *parameter*

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments**

| Name | Instance | Encoding |
|---|---|---|
| *parameter* | *Continuous Current Limit* | 0 |
| | *Energy Limit* | 1 |

| | Type | Range/Scaling |
|---|---|---|
| *value* | unsigned 16-bit | see below |

**Packet Structure**

**SetCurrentFoldback**

| | 15–8 | 7–0 |
|---|---|---|
| | checksum | **41**h |

| | First data word (15–0) |
|---|---|
| write | *parameter* |

| | Second data word (15–0) |
|---|---|
| write | *value* |

**GetCurrentFoldback**

| | 15–8 | 7–0 |
|---|---|---|
| | checksum | **42**h |

| | First data word (15–0) |
|---|---|
| write | *parameter* |

| | Second data word (15–0) |
|---|---|
| read | *value* |

**Description**

**SetCurrentFoldback** is used to set various $I^2t$ foldback-related parameters. Two parameters can be set, the *Continuous Current Limit*, and the *Energy Limit*. The units of *Continuous Current Limit* are convertible to milliAmps, and represent percentage of maximum peak current, with scaling of 100/2^15. The range is from 0% to the factory default continuous current limit setting.

The maximum current is the largest current that can be represented rather than the maximum that can be sourced or sensed. The maximum current can be calculated via the formula

*Max = Current Scaling * 0x8000*

For example for the high power Altas, using the scale factor from Section 3.11, “Atlas Conversion Factors”, the maximum current = 1.526mA * 0x8000 = 50A.

The units of *Energy Limit* are convertible to $Amp^2$Seconds, and represent the percentage of maximum energy, with scaling of 100/2^15. The range is from 0% to the factory default energy limit setting.

**Description (cont.)**

The *Continuous Current Limit* is used by the current foldback algorithm. When the current output of the drive exceeds this setting, accumulation of the $I^2$ energy above this setting begins. Once the accumulated excess $I^2$ energy exceeds the value specified by the *Energy Limit* parameter, a current foldback condition exists and the commanded current will be limited to the specified *Continuous Current Limit*. When this occurs, the Current Foldback bit in the Event Status and Drive Status registers will be set. When the accumulated $I^2$ energy above the *Continuous Current Limit* drops to zero (0), the limit is removed, and the Current Foldback bit in the Drive Status register is cleared.

**SetEventAction** can be used to configure a change in operating mode when current foldback occurs. Doing this does not interfere with the basic operation of Current Foldback described above. If this is done, the Current Foldback bit in the Event Status register must be cleared prior to restoring the operating mode, regardless of whether the system is in current foldback or not.

When current control is not active, a current foldback event always causes a change to the disabled state (all loops and motor output are disabled), regardless of the programmed Event Action. Changing the operating mode from disabled requires clearing of the Current Foldback bit in Event Status.

**GetCurrentFoldback** gets the maximum continuous current setting.

**Restrictions**

Values of *Continuous Current Limit* greater than the factory setting for maximum continuous current are not allowed.

**see**

**GetEventStatus** (p. 106), **ResetEventStatus** (p. 124), **GetDriveStatus** (p. 105), **RestoreOperatingMode** (p. 125), **GetActiveOperatingMode** (p. 98)

# SetCurrentLoop buffered 73h
# GetCurrentLoop 74h



**Syntax**

**SetCurrentLoop** *phase parameter value*
**GetCurrentLoop** *phase parameter*

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments**

| Name | Instance | Encoding |
|---|---|---|
| *phase* | *Phase A* | 0 |
| | *Phase B* | 1 |
| | *Both (A and B)* | 2 |
| *parameter* | *Proportional Gain (KpCurrent)* | 0 |
| | *Integral Gain (KiCurrent)* | 1 |
| | *Integral Sum Limit (ILimitCurrent)* | 2 |
| | **Type** | **Range/Scaling** |
| *value* | unsigned 16 bits | see below |

**Packet Structure**

**SetCurrentLoop**

| | 15–8 | 7–0 |
|---|---|---|
| | checksum | **73**h |

First data word

| | 15–12 | 11–8 | 7–0 |
|---|---|---|---|
| write | 0 | *phase* | *parameter* |

Second data word

| | 15–0 |
|---|---|
| write | *value* |

**GetCurrentLoop**

| | 15–8 | 7–0 |
|---|---|---|
| | checksum | **74**h |

First data word

| | 15–12 | 11–8 | 7–0 |
|---|---|---|---|
| write | 0 | *phase* | *parameter* |

Second data word

| | 15–0 |
|---|---|
| read | *value* |

**Description**

**Set/GetCurrentLoop** is used to configure the operating parameters of the A/B current loops. See Section 4.5, "Current Loop" for more information on how each ***parameter*** is used in the current loop processing. The ***value*** written/read is always an unsigned 16-bit value, with the parameter-specific scaling shown below:

| Parameter | Range | Scaling | Units |
|---|---|---|---|
| *Proportional Gain (KpCurrent)* | 0 to 2^15–1 | 1/64 | gain |
| *Integral Gain (KiCurrent)* | 0 to 2^15–1 | 1/256 | gain/cycles |
| *Integral Sum Limit (ILimitCurrent)* | 0 to 2^15–1 | 1/100 | % current * cycles |

A setting of 64 for *KpCurrent* corresponds to a gain of 1. That is, an error signal of 30% maximum current will cause the proportional contribution of the current loop output to be 30% of maximum output. Similarly, setting *KiCurrent* to 256 gives it a gain of 1, and the value of the integral sum would become the integral contribution to the output. The units of time for the integral sum are cycles.

**Description (cont.)**

*ILimitCurrent* is used to limit the contribution of the integral sum at the output. Its effect depends on the value of *KiCurrent*. Setting *ILimitCurrent* to 1000 when *KiCurrent* is 10 means that the maximum contribution to the output is 1000 x 10 = 10,000 out of 2^15 - 1 or approximately 30.5%

The *phase* argument can be used to set the operating parameters for the A and B loops independently. In most cases, the A and B loops will not require different operating parameters, so **SetCurrentLoop** can be used with a *phase* of 2, which sets both the A and B loops in a single API command. For **GetCurrentLoop**, a *phase* of 2 is not valid.

**Restrictions**

**Set/GetCurrentLoop** are buffered commands. All parameters set are buffered, and will not take effect until an update is done on the current loop by setting the update bit in a torque command. The values read by **GetCurrentLoop** are the buffered settings.

This command is only supported in products that include digital current control, and when the current control mode is A/B.

**see**

**GetCurrentLoopValue** (p. 102), **Set/GetCurrentControlMode** (p. 130)

# SetDriveCommandMode 7Eh
# GetDriveCommandMode 7Fh

## Syntax

**SetDriveCommandMode** *mode*
**GetDriveCommandMode**

## Arguments

| Name | Instance | Encoding |
|---|---|---|
| *format* | *BLDC* | 0 |
| | *Step* | 4 |
| | *DC* | 7 |
| transport | *SPI* | 0 |
| | *Pulse and Direction* | 1 |

## Packet Structure

**SetDriveCommand mode**

| | 15–8 | 7–0 |
|---|---|---|
| write | checksum | **7E**h |
| write | *transport* (15) | *format* (0) |

**GetDriveCommand mode**

| | 15–8 | 7–0 |
|---|---|---|
| write | checksum | **7F**h |
| write | *transport* (15) | *format* (0) |

## Description

**SetDriveCommandMode** may be used to change the means of commanding Atlas motor torque. The transport field specifies the physical interface used for torque commands, either the default Serial Peripheral Interface, or Pulse and Direction. See Section 4.9.1, "Pulse & Direction Signal Input Mode" for more information on pulse and direction mode.

The format field specifies the layout of the torque command; in the Atlas only one format is supported for each motor type. See Section 5.3, "Sending a Voltage or Torque Output Value" for details.

The format for pulse and direction input must be Step.

# SetDriveFaultParameter 62h
# GetDriveFaultParameter 60h

**Syntax**

**SetDriveFaultParameter** *parameter value*
**GetDriveFaultParameter** *parameter*

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments**

| Name | Instance | Encoding |
|---|---|---|
| *parameter* | *Overvoltage limit* | 0 |
| | *Undervoltage limit* | 1 |
| | *Recovery mode* | 2 |
| | *Watchdog limit* | 3 |
| | *Temperature limit* | 4 |
| | *Temperature hysteresis* | 5 |

| | Type | Range/Scaling |
|---|---|---|
| *value* | unsigned 16 bits | see below |

**Packet Structure**

**SetDriveFaultParameter**

| | 15 … 8 | 7 … 0 |
|---|---|---|
| write | checksum | **62**h |
| write | *parameter* (15 … 0) | |
| write | *value* (15 … 0) | |

**GetDriveFaultParameter**

| | 15 … 8 | 7 … 0 |
|---|---|---|
| write | checksum | **64**h |
| write | *parameter* (15 … 0) | |
| read | *value* (15 … 0) | |

**Description**

**SetDriveFaultParameter** is used to set various operating limits and parameters affecting drive fault handling. Several aspects of fault handling may be controlled:

- Bus voltage fault: *Overvoltage limit*, *Undervoltage limit*
- How fault and event recovery is handled: *Recovery mode*
- Communication watchdog: *Watchdog limit*
- Overtemperature fault: *Temperature limit*, *Temperature hysteresis*

The overvoltage and undervoltage limits are limits on bus voltage, and use the same scaling as the **GetBusVoltage** command. Whenever bus voltage is greater than the overvoltage limit an overvoltage drive fault will be signaled, if bus voltage is less than the undervoltage limit an undervoltage drive fault will be signaled. In either case a drive exception event will be raised and motor output disabled.

The watchdog limit specifies the number of current loop cycles that may elapse without receiving a valid torque channel command from a controlling processor. If this limit is exceeded then a watchdog timeout fault will be signaled and a drive exception event raised, disabling motor output. The specified value is scaled by 8. For example, to specify that a watchdog timeout should occur after 16 cycles then the value should be 2. A value of zero disables the watchdog timeout.

## Description (cont.)

The temperature limit specifies a temperature above which an overtemperature drive fault is signaled and an overtemperature event raised, disabling motor output. After an overtemperature event the temperature must fall to the temperature limit minus the hysteresis value before the event and drive fault may be cleared.

Both the overtemperature limit and the temperature hysteresis use the same scaling as the **GetTemperature** command.

The recovery mode is an enumerated value, either commanded (0), or automatic (1). In the default commanded mode an explicit **ResetEventStatus** command is required after a disabling event to clear the responsible event bits. A **ClearDriveFaultStatus** command is not required, but is recommended to clear all drive fault status bits. In the automatic recovery mode Atlas will internally clear the event and drive fault status registers and attempt to automatically re-enable the commanded operating mode after a disabling event has occurred. Before automatic recovery will be attempted the controlling processor must acknowledge the fault by first driving the ~Enable line high, and then low. It is highly recommended that the **SetDriveFaultOutMask** command be used to trigger the *FaultOut* signal on any drive faults that will result in output being disabled, so that a controlling processor may recognize the fault.

In the commanded recovery mode the **SetOperatingMode** command will signal an error if attempting to enable output when any disabling conditions are present, and the commanded operating mode will not be changed. In the automatic recovery mode the **SetOperatingMode** command will not fail under these circumstances, but will change the command (but not the active) operating mode.

## Restrictions

## see

**GetBusVoltage** (p. 99), **GetTemperature** (p. 114), **ResetEventStatus** (p. 124), **ClearDriveFaultStatus** (p. 94), **SetFaultOutMask** (p. 140), **SetOperatingMode** (p. 144)

# SetDrivePWM 23h
# GetDrivePWM 24h

**Syntax**

**SetDrivePWM** *option value*
**GetDrivePWM** *option*

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments**

| Name | Instance | | | Encoding |
|---|---|---|---|---|
| *option* | *Limit* | | | 0 |
| | **Type** | **Range** | **Scaling** | **Units** |
| *value* | 16-bit unsigned | 0 *to* $2^{14}$–1 | $100/2^{14}$ | % max output |

**Packet Structure**

**SetDrivePWM**

| | Bits 15–8 | Bits 7–0 |
|---|---|---|
| write | checksum | **23**h |
| write | *option* (15–0) | |
| write | *value* (15–0) | |

**GetDrivePWM**

| | Bits 15–8 | Bits 7–0 |
|---|---|---|
| write | checksum | **24**h |
| write | *option* (15–0) | |
| read | *value* (15–0) | |

**Description**

The **SetDrivePWM** command is used to set the PWMLimit register, which limits the maximum PWM duty cycle, and hence the effective output voltage. The PWM limit is applied to each terminal individually, whether the current loop is enabled or not. The **GetDrivePWM** returns the current value of the PWMLimit register.

**Restrictions**

## Syntax

**SetEventAction** *event action*
**GetEventAction** *event*

## Motor Types

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

## Arguments

| Name | Instance | Encoding |
|---|---|---|
| *event* | *Immediate* | 0 |
| | — (Reserved) | 1-3 |
| | *Current Foldback* | 4 |
| *action* | *None* | 0 |
| | — (Reserved) | 1-6 |
| | *Disable Motor Output & Higher Modules* | 7 |

## Packet Structure

**SetEventAction**

| | Bits 15–8 | Bits 7–0 |
|---|---|---|
| | checksum | **48**h |
| write (First data word, bits 15–0) | *event* | |
| write (Second data word, bits 15–0) | *action* | |

**GetEventAction**

| | Bits 15–8 | Bits 7–0 |
|---|---|---|
| | checksum | **49**h |
| write (First data word, bits 15–0) | *event* | |
| read (Second data word, bits 15–0) | *action* | |

## Description

**SetEventAction** configures what actions will be taken by Atlas in response to a given *event*. The *action* can be either to modify the operating mode by disabling wafer output, or to take no action.

When, through **SetEventAction**, one of the *events* causes an *action*, the event bit in the Event Status register must be cleared prior to returning to operation. For changes in operating mode, this means that the bit must be cleared prior to restoring the operating mode, either by **RestoreOperatingMode** or **SetOperatingMode**.

**GetEventAction** gets the action that is currently programmed for the given event with the exception of the *Immediate* event, which cannot be read back.

## Restrictions

## see

**GetActiveOperatingMode** (p. 98), **RestoreOperatingMode** (p. 125), **Set/GetOperatingMode** (p. 144)

# SetFaultOutMask FBh
# GetFaultOutMask FCh

**Syntax**

**SetFaultOutMask** *mask*
**GetFaultOutMask**

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments**

| Name | Instance | Encoding |
|---|---|---|
| *mask* | see below | bitmask |

**Packet Structure**

**SetFaultOutMask**

| | checksum (15–8) | FBh (7–0) |
|---|---|---|

First data word

| write | *mask* (15–0) |
|---|---|

**GetFaultOutMask**

| | checksum (15–8) | FCh (7–0) |
|---|---|---|

First data word

| read | *mask* (15–0) |
|---|---|

**Description**

**SetFaultOutMask** configures the mask on Drive Fault Status register bits that will be ORed together on the FaultOut pin. The FaultOut pin is active high, as are the bits in Drive Fault Status. Thus, FaultOut will go high when any of the enabled bits in Drive Fault Status are set (1). The *mask* parameter is used to determine what bits in the Drive Fault Status register can cause FaultOut high, as follows:

| Name | Bit |
|---|---|
| Overcurrent | 0 |
| — (Reserved) | 1-2 |
| Operating Mode Mismatch | 3 |
| — (Reserved) | 4 |
| Overvoltage | 5 |
| Undervoltage | 6 |
| Disabled | 7 |
| Current Foldback | 8 |
| Overtemperature | 9 |
| SPI Checksum Error | 10 |
| Watchdog Timeout | 11 |
| — (Reserved) | 12-15 |

For example, a *mask* setting of hexadecimal 2060h will configure the FaultOut pin to go high upon Overtemperature Fault, or Bus Voltage Fault. The FaultOut pin stays high until all Fault enabled bits in Event Status are cleared. The default value for the FaultOut *mask* is 71h.

**GetFaultOutMask** gets the current *mask*.

**Restrictions**

.

**see**

**GetDriveFaultStatus** (p. 103), **ClearDriveFaultStatus** (p. 94)

## Syntax

**SetFOC** *loop parameter value*
**GetFOC** *loop parameter*

## Motor Types

| | Brushless DC | Microstepping |
|---|---|---|

## Arguments

| Name | Instance | Encoding |
|---|---|---|
| *loop* | *Direct(d)* | 0 |
| | *Quadrature(q)* | 1 |
| | *Both(d and q)* | 2 |
| *parameter* | *Proportional Gain (KpDQ)* | 0 |
| | *Integral Gain (KiDQ)* | 1 |
| | *Integral Sum Limit (ILimitDQ)* | 2 |
| | **Type** | **Range/Scaling** |
| *value* | unsigned 16 bits | see below |

## Packet Structure

**SetFOC**

| checksum (15–8) | **F6h** (7–0) |
|---|---|

First data word

| | 0 (15–12) | *loop* (11–8) | *parameter* (7–0) |
|---|---|---|---|
| write | 0 | *loop* | *parameter* |

Second data word

| | *value* (15–0) |
|---|---|
| write | *value* |

**GetFOC**

| checksum (15–8) | **F7h** (7–0) |
|---|---|

First data word

| | 0 (15–12) | *loop* (11–8) | *parameter* (7–0) |
|---|---|---|---|
| write | 0 | *loop* | *parameter* |

Second data word

| | *value* (15–0) |
|---|---|
| read | *value* |

## Description

**Set/GetFOC** is used to configure the operating parameters of the FOC-Current control. See Section 4.5, "Current Loop" for more information on how each ***parameter*** is used in the current loop processing. The ***value*** written/read is always an unsigned 16-bit value, with the parameter-specific scaling shown below:

| Parameter | Range | Scaling | Units |
|---|---|---|---|
| *Proportional Gain (KpDQ)* | 0 to 2^15–1 | 1/64 | gain |
| *Integral Gain (KiDQ)* | 0 to 2^15–1 | 1/256 | gain/cycles |
| *Integral Sum Limit (ILimitDQ)* | 0 to 2^15–1 | 1/100 | % current * cycles |

A setting of 64 for ***KpDQ*** corresponds to a gain of 1. That is, an error signal of 30% maximum current will cause the proportional contribution of the current loop output to be 30% of maximum output.

**Description (cont.)**

Similarly, setting *KiDQ* to 256 gives it a gain of 1; the value of the integral sum would become the integral contribution to the output.

*ILimitDQ* is used to limit the contribution of the integral sum at the output. Its effect depends on the value of *KiDQ*. Setting *IlimitDQ* to 1000 when *KiDQ* is 10 means that the maximum contribution to the output is 1000 x 10 = 10,000 out of 2^15 - 1 or approximately 30.5%. The units of time for the integral sum are cycles.

The *loop* argument allows individual configuration of the parameters for the D and Q current loops. Alternately, a *loop* of 2 can be used with **SetFOC** to set the D and Q loops with a single API command. A *loop* of 2 is not valid for **GetFOC**.

**Restrictions**

**Set/GetFOC** are buffered commands. All parameters set are buffered, and will not take effect until an update is done on the current loop by sending a torque command with the **Update** bit set. The values read by **GetFOC** are the buffered settings.

These commands are only supported in products that include digital current control, and when the current control mode is set to FOC.

**see**

**GetFOCValue** (p. 107), **Set/GetCurrentControlMode** (p. 130)

# SetMotorType 03h
# GetMotorType 03h

## Syntax

**SetMotorType** *axis type*
**GetMotorType** *axis*

## Motor Types

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

## Arguments

| Name | Instance | Encoding |
|---|---|---|
| *type* | Brushless DC (3 phase) | 0 |
| | Microstepping (2 phase) | 3 |
| | DC Brush | 7 |

## Packet Structure

**SetMotorType**

| checksum (bits 15–8) | **02**h (bits 7–0) |
|---|---|

Data

| | bits 15–3 | bits 2–0 |
|---|---|---|
| write | (shaded) | *type* |

**GetMotorType**

| checksum (bits 15–8) | **03**h (bits 7–0) |
|---|---|

Data

| | bits 15–3 | bits 2–0 |
|---|---|---|
| read | (shaded) | *type* |

## Description

**SetMotorType** sets type of motor being driven by Atlas. This operation sets the number of phases for commutation on the axis, as well as internally configuring the motion control IC for the motor type.

The following table describes each motor type, and the number of phases to be commutated.

| Motor type | Commutation |
|---|---|
| Brushless DC (3 phase) | 3 phase |
| Microstepping (2 phase) | 2 phase |
| DC Brush | None |

**GetMotorType** returns the configured motor type for the selected *axis*.

## Restrictions

The motor type should only be set once for each axis, either via NVRAM initialization during device startup, or immediately after reset using **SetMotorType**. Once it has been set, it should not be changed. Executing **SetMotorType** will reset many variables to their motor type specific default values.

Only multimotor Atlases accept the **SetMotorType** command. When using multimotor Atlas with Magellan separate **SetMotorType** commands must be sent, first to Atlas, then to Magellan.

## C-Motion API

```
PMDResult PMDSetMotorType (PMDAxisInterface axis_intf, PMDuint8 type)
PMDResult PMDGetMotorType (PMDAxisInterface axis_intf, PMDuint8* type)
```

# SetOperatingMode 65h
# GetOperatingMode 66h

**Syntax**

**SetOperatingMode** *mode*
**GetOperatingMode**

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments**

| Name | Instance | Encoding |
|---|---|---|
| | **Type** | **Range/Scaling** |
| *mode* | unsigned 16-bit | see below |

**Packet Structure**

**SetOperatingMode**

| checksum (15–8) | **65**h (7–0) |
|---|---|

write — First data word

| *mode* (15–0) |
|---|

**GetOperatingMode**

| checksum (15–8) | **66**h (7–0) |
|---|---|

read — First data word

| *mode* (15–0) |
|---|

**Description**

**SetOperatingMode** configures the operating mode. Each bit of the *mode* configures whether a feature/loop is active or disabled, as follows:

| Name | Bit | Description |
|---|---|---|
| — | 0 | Reserved |
| Motor Output Enabled | 1 | 0: motor outputs disabled. 1: motor outputs enabled. |
| Current Control Enabled | 2 | 0: current control bypassed. 1: current control active. |
| — | 3-15 | Reserved |

When the current loop is disabled, it operates by passing its input directly to its output, and clearing all internal state variables (such as integrator sums, etc.).

For example, to configure Atlas for current loop operation the operating mode would be set to hexadecimal 0007h.

When Atlas is in the default event recovery mode SetOperatingMode will signal an error if an event that would disable motor output is present, and will not change any internal registers. The Invalid Operating Mode Restore error (10h) error will be signaled.

An alternative event recovery mode (automatic recovery mode) may be set by using the **SetDriveFaultParameter** command, in this mode **SetOperatingMode** will not signal an error if a disabling event is present, but will instead change only the static operating mode, and leave the active operating mode as it is. For this reason the event recovery mode should be set first in during user initialization from nonvolatile memory, and the operating mode set afterwards.

**Description (cont.)**

This command should be used to configure the static operating mode. The actual current operating mode may be changed in response to safety events, or user-programmable events. In this case, the present operating mode is available using **GetActiveOperatingMode**. **GetOperatingMode** will always return the static operating mode set using **SetOperatingMode**. Executing the **SetOperatingMode** command sets both the static operating mode and the active operating mode to the desired state.

**GetOperatingMode** gets the static operating mode.

**Restrictions**

**see**

**GetActiveOperatingMode** (p. 98), **RestoreOperatingMode** (p. 125), **GetDriveFaultStatus** (p. 103)

# SetPhaseCounts 75h
# GetPhaseCounts 7Dh

**Syntax**

**SetPhaseCounts** *counts*
**GetPhaseCounts**

**Motor Types**

| | Microstepping |
|---|---|

**Arguments**

| Name | Instance | Encoding | | |
|---|---|---|---|---|
| | Type | Range | Scaling | Units |
| *counts* | unsigned 16 bits | 1 *to* 1024 | unity | microsteps |

**Packet Structure**

**SetPhaseCounts**

| checksum (15–8) | **75**h (7–0) |
|---|---|

Data

| write | *counts* (15–0) |
|---|---|

**GetPhaseCounts**

| checksum (15–8) | **7D**h (7–0) |
|---|---|

Data

| read | *counts* (15–0) |
|---|---|

**Description**

For axes configured for microstepping motor types, the number of microsteps per full step is set using the **SetPhaseCounts** command. The parameter used for this command represents the number of microsteps per electrical cycle (4 times the desired number of microsteps). For example, to set 64 microsteps per full step, the **SetPhaseCounts 256** command should be used. The maximum number of microsteps that can be generated per full step is 256, giving a maximum parameter value of 1024.

**GetPhaseCounts** returns the number of counts or microsteps per electrical cycle.

**Restrictions**

This command is useful only when driving miscrostepping motors in pulse and direction mode.

**see**

# SetPWMFrequency 0Ch
# GetPWMFrequency 0Dh

**Syntax** **SetPWMFrequency** *frequency*
**GetPWMFrequency**

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments**

| Name | Instance | Encoding | | |
|---|---|---|---|---|
| | Type | Range | Scaling | Units |
| *frequency* | unsigned 16 bits | see below | 125/32 | Hz |

**Packet Structure**

**SetPWMFrequency**

| checksum (15–8) | **0C**h (7–0) |
|---|---|

Data

| write | *frequency* (15–0) |
|---|---|

**GetPWMFrequency**

| checksum (15–8) | **0D**h (7–0) |
|---|---|

Data

| read | *frequency* (15–0) |
|---|---|

**Description**

**SetPWMFrequency** sets the PWM output frequency (in kHz). To select one of the supported frequencies, pass the value listed in the SetPWMFrequency Value column as the *frequency* argument to this command. If the frequency specified is not supported the argument will be sampled down.

| Approximate Frequency | PWM bit Resolution | Actual Frequency | SetPWMFrequency Value |
|---|---|---|---|
| 20 kHz | 10 | 19.531 kHz | 5,000 |
| 40 kHz | 9 | 39.062 kHz | 10,000 |
| 80 kHz | 8 | 78.124 kHz | 20,000 |
| 120 kHz | 7 | 117.187 kHz | 30,000 |

**Restrictions**

The PWM frequency can be changed only when motor output is disabled (e.g., immediately after power-up or reset).

**see**

## Syntax

**SetTraceMode** *mode*
**GetTraceMode**

## Motor Types

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

## Arguments

| Name | Instance | Encoding |
|---|---|---|
| *roll mode* | *One Time* | 0 |
| | *Rolling Buffer* | 1 |
| *Trigger* | *Internal Trigger* | 0 |
| | *External Trigger* | 1 |

## Packet Structure

**SetTraceMode**

| | 15–8 | 7–0 |
|---|---|---|
| write | checksum | **B0**h |

Data

| | 15–8 | 7 | 6–1 | 0 |
|---|---|---|---|---|
| write | 0 | Trigger | 0 | *roll mode* |

**GetTraceMode**

| | 15–8 | 7–0 |
|---|---|---|
| write | checksum | **B1**h |

Data

| | 15–8 | 7 | 6–1 | 0 |
|---|---|---|---|---|
| read | 0 | Trigger | 0 | *roll mode* |

## Description

**SetTraceMode** sets the buffer usage for the next trace. In *One Time* mode, the trace continues until the trace buffer is filled, then stops. In *Rolling Buffer* mode, the trace continues from the beginning of the trace buffer after the end is reached. When in the rolling mode, values stored at the beginning of the trace buffer are lost if they are not read before being overwritten by the wrapped data.

The Roll Mode controls buffer usage. In One Time mode, the trace continues until the trace buffer is filled, then stops. In Rolling Buffer mode, the trace continues writing at the beginning of the buffer after the end is reached. When in the rolling mode, values stored at the beginning of the trace buffer are lost if they are not read before being overwritten by the wrapped data.

The Trigger bit controls the timing of trace samples. In Internal Trigger mode the trace timing is controlled by the TracePeriod register and Atlas internal clock; a trace capture is done every TracePeriod. In External Trigger mode the trace timing is controlled by the trace bit in the SPI torque command; exactly one trace capture is done immediately after a set trace bit is received, so that trace timing is controlled externally. Internal trigger mode gives lower jitter and a higher possible trace sampling rate, but external trigger mode allows approximate synchronization with an external clock.

**GetTraceMode** returns the value for the trace mode.

## Restrictions

## see

**GetTraceStatus** (p. 117), **Set/GetTracePeriod** (p. 149), **Set/GetTraceStart** (p. 150), **Set/GetTraceStop** (p. 151),

# SetTracePeriod B8h
# GetTracePeriod B9h



**Syntax**

**SetTracePeriod** *period*
**GetTracePeriod**

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments**

| Name | Type | Range | Scaling | Units |
|---|---|---|---|---|
| *period* | unsigned 16 bits | 1 *to* $2^{16}$–1 | unity | cycles |

**Packet Structure**

**SetTracePeriod**

| | 15 … 8 | 7 … 0 |
|---|---|---|
| write | checksum | **B8**h |

Data

| | 15 … 0 |
|---|---|
| write | *period* |

**GetTracePeriod**

| | 15 … 8 | 7 … 0 |
|---|---|---|
| write | checksum | **B9**h |

Data

| | 15 … 0 |
|---|---|
| read | *period* |

**Description**

**SetTracePeriod** sets the interval between contiguous trace captures. For example, if the trace period is set to one, trace data will be captured at the end of every chip cycle. If the trace period is set to two, trace data will be captured at the end of every second chip cycle, and so on.

**GetTracePeriod** returns the value for the trace period.

**Restrictions**

The trace period is used only in Internal Trigger mode.

**see**

**Set/Get Trace Mode** (p. 148)

# SetTraceStart B2h
# GetTraceStart B3h

**Syntax**

**SetTraceStart** *condition*
**GetTraceStart**

**Arguments**

| Name | Instance | Encoding |
|---|---|---|
| *condition* | *Immediate* | 0 |
| | *SPI Command* | 6 |

**Packet Structure**

**SetTraceStart**

| | checksum | **B2**h |
|---|---|---|
| write | 15 … 8 | 7 … 0 |

Data

| 0 | condition | 0 |
|---|---|---|
| 15 … 8 | 7 … 4 | 3 … 0 |

**GetTraceStart**

| | checksum | **B3**h |
|---|---|---|
| write | 15 … 8 | 7 … 0 |

Data

| 0 | condition | 0 |
|---|---|---|
| 15 … 8 | 7 … 4 | 3 … 0 |

**Description**

**SetTraceStart** sets the condition for starting a trace, and must be called before any tracing can be done. If the immediate condition is specified tracing will begin as soon as the command is processed. If the SPI Command condition is specified tracing will begin after an SPI command header is received with the trace bit set.

**GetTraceStart** returns the value of the trace-start trigger.

Once a trace is started the trace-start trigger is reset to zero.

The trace start condition is not checked if trace is already running.

When using External Trigger mode no trace samples will be captured unless the SPI command header trace bit is set.

**see**

**Set/GetBufferLength** (p. 126), **Set/GetTracePeriod** (p. 149), **Set/GetTraceMode** (p. 148), **Set/GetTraceStop** (p. 151), **GetTraceCount** (p. 116), **GetTraceStatus** (p. 117)

# SetTraceStop B4h
# GetTraceStop B5h



**Syntax**

**SetTraceStop** *condition*
**GetTraceStop**

**Arguments**

| **Name** | **Instance** | **Encoding** |
|---|---|---|
| *condition* | *Immediate* | 0 |
| | *SPI Command* | 6 |

**Packet Structure**

**SetTraceStop**

| | 15–8 | 7–0 |
|---|---|---|
| write | checksum | **B4**h |

Data

| 15–8 | 7–4 | 3–0 |
|---|---|---|
| 0 | condition | 0 |

**GetTraceStop**

| | 15–8 | 7–0 |
|---|---|---|
| write | checksum | **B5**h |

Data

| 15–8 | 7–4 | 3–0 |
|---|---|---|
| 0 | condition | 0 |

**Description**

**SetTraceStop** sets the condition for starting a trace, and must be called to reset the internal trace state before starting again. If the immediate condition is specified tracing will stop as soon as the command is processed. If the SPI Command condition is specified tracing will stop after an SPI command header is received with the trace bit clear.

**GetTraceStop** returns the value of the trace-stop trigger.

Once a trace is stopped the trace-start trigger is reset to zero.

The trace stop condition is not checked if trace is not running.

**see**

**Set/GetBufferLength** (p. 126), **Set/GetTracePeriod** (p. 149), **Set/GetTraceMode** (p. 148), **Set/GetTraceStart** (p. 150), **GetTraceCount** (p. 116), **GetTraceStatus** (p. 117)

# SetTraceVariable B6h
# GetTraceVariable B7h

**Syntax**

**SetTraceVariable** *variableNumber variableID*
**GetTraceVariable** *variableNumber*

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments**

| Name | Instance | Encoding |
|---|---|---|
| *variableNumber* | *Variable1* | 0 |
| | *Variable2* | 1 |
| | *Variable3* | 2 |
| | *Variable4* | 3 |
| *variableID* | | |
| *Status Registers* | | |
| | *Event Status* | 12 |
| | *Signal Status* | 14 |
| | *Drive Status* | 56 |
| | *Drive Fault Status* | 79 |
| | *SPI Status* | 80 |
| *Commutation/Phasing* | *Active Motor Command* | 7 |
| | *Phase A Command* | 17 |
| | *Phase B Command* | 18 |
| | *Phase C Command* | 19 |
| | *Phase Angle Scaled* | 29 |
| *Current Loops* | *Phase A Reference* | 66 |
| | *Phase B Reference* | 67 |
| | *Phase A Error* | 30 |
| | *Phase B Error* | 35 |
| | *Phase A Actual Current* | 31 |
| | *Phase B Actual Current* | 36 |
| | *Phase A Integral Contribution* | 33 |
| | *Phase B Integral Contribution* | 38 |
| | *Current Loop A Output* | 34 |
| | *Current Loop B Output* | 39 |
| *Field Oriented Control* | *D Reference* | 40 |
| | *Q Reference* | 46 |
| | *D Error* | 41 |
| | *Q Error* | 47 |
| | *D Feedback* | 42 |
| | *Q Feedback* | 48 |
| | *D Integrator Contribution* | 44 |
| | *Q Integrator Contribution* | 50 |
| | *D Output* | 45 |
| | *Q Output* | 51 |
| | FOC *Phase A* | 52 |
| | FOC *Phase B* | 53 |
| | *Alpha Output* | 73 |
| | *Beta Output* | 74 |
| | *Phase A Actual Current* | 31 |
| | *Phase B Actual Current* | 36 |

## Arguments (cont.)

| | | |
|---|---|---|
| *Motor Output* | *Bus Voltage* | 54 |
| | *Temperature* | 55 |
| | *I2t Energy* | 68 |
| | *Terminal A Output* | 75 |
| | *Terminal B Output* | 76 |
| | *Terminal C Output* | 77 |
| | *Leg Current A* | 69 |
| | *Leg Current B* | 70 |
| | *Leg Current C* | 71 |
| | *Leg Current D* | 72 |
| | *Clip Factor* | 78 |
| *Miscellaneous* | *None* (disable variable) | 0 |
| | *Atlas Time* | 8 |

## Packet Structure

**SetTraceVariable**

| | Bits 15–8 | Bits 7–0 |
|---|---|---|
| write | checksum | **B6**h |

First data word

| | Bits 15–2 | Bits 1–0 |
|---|---|---|
| write | 0 | *variableNumber* |

Second data word

| | Bits 15–8 | Bits 7–0 |
|---|---|---|
| write | *variableID* | 0 |

**GetTraceVariable**

| | Bits 15–8 | Bits 7–0 |
|---|---|---|
| write | checksum | **B7**h |

First data word

| | Bits 15–2 | Bits 1–0 |
|---|---|---|
| write | 0 | *variableNumber* |

Second data word

| | Bits 15–8 | Bits 7–0 |
|---|---|---|
| read | *variableID* | 0 |

## Description

**SetTraceVariable** assigns the given variable to the specified ***variableNumber*** location in the trace buffer. Up to four variables may be traced at one time.

All variable assignments must be contiguous starting with ***variableNumber*** = 0.

**GetTraceVariable** returns the variable of the specified ***variableNumber***.

**Example:** To set up a three variable trace capturing the motor command, phase angle, and phase A current, the following commands would be used:

SetTraceVariable 0 7
SetTraceVariable 1 29
SetTraceVariable 2 31
SetTraceVariable 3 0

## Restrictions

When retrieving the traced variables from the list above the scaling may depend on the type of Atlas being used. See Section 3.11.1, "Atlas Settings Defaults and Limits" for scaling information. See also Figure 4-6, Figure 4-7 and Figure 4-8 for current loop, field oriented control, and motor output related variable scaling.

## see

**SetTracePeriod** (p. 149)

# Update

**Syntax** **Update**

**Motor Types**

| DC Brush | Brushless DC | Microstepping |
|---|---|---|

**Arguments** None

**Packet Structure**

| Update | |
|---|---|
| checksum | **1A**h |
| 15 8 | 7 0 |

**Description** The **Update** command should only be called from the nonvolatile initialization memory, when using SPI communication the update bit in the torque command should be set to command an update.

**Restrictions** Non-volatile initialization only.

**see**

# A. Atlas Developer Kits

A

## *In This Appendix*



## A.1 Overview

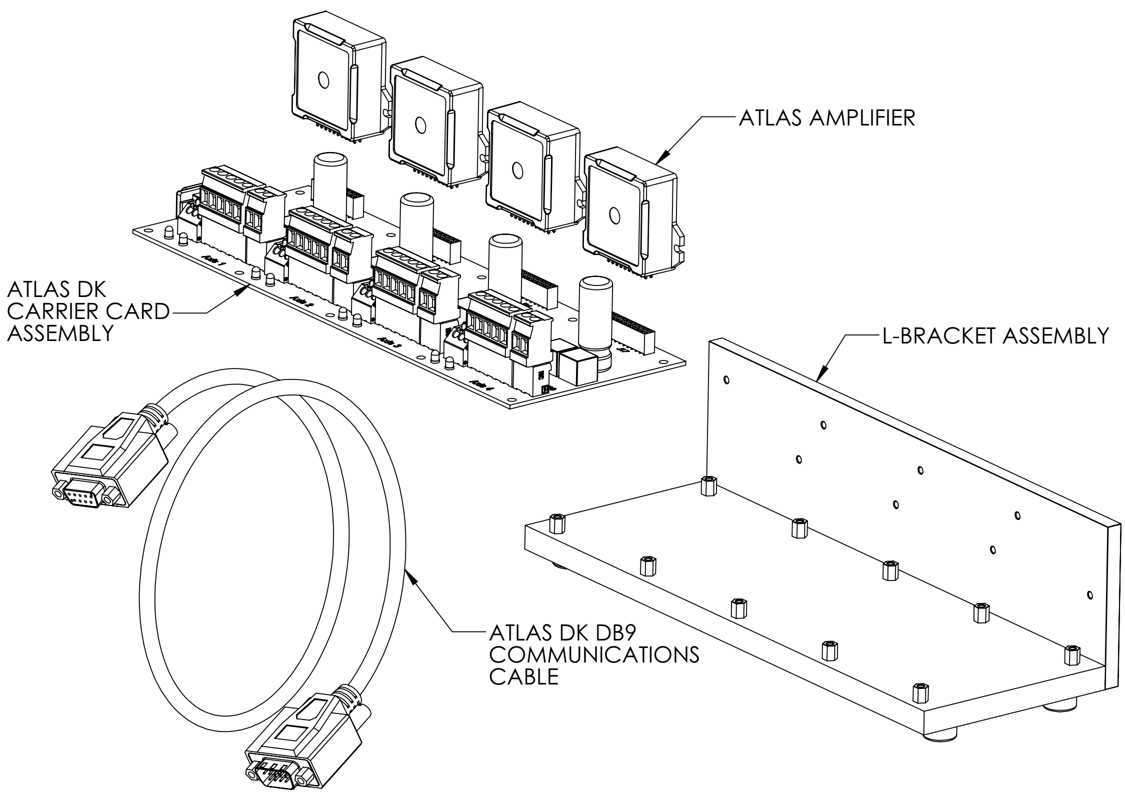



**Figure A-1: Developer Kit Components (four-axis version shown)**

To simplify development with Atlas Amplifiers several developer kits (DK) are available.

The major elements of the DK are:

- Atlas DK board (comes in 1 or 4 axis version)
- Atlas DK DB9 communications cable

- Base plate and, if vertical Atlas DKs are ordered, vertical plate forming an L-bracket for heat sink attachment with associated mounting hardware (comes in 1 or 4 axis version).
- Compact to ultra-compact size Atlas converter cards.
- For horizontal DKs heat sinks with adhesive thermal pads
- Various other assembly components such as screws and allen keys depending on the Atlas DK type ordered

# A.2 Developer Kit P/Ns

There are four available Atlas developer kits, reflecting a choice of one or four axis board, and a choice of two different Atlas mounting configurations; vertical and horizontal. The following table shows this:

| Developer Kit P/N | # of Axes | Atlas Type |
|---|---|---|
| MDK1LI0000V | 1 | Vertical |
| MDK1LI0000H | 1 | Horizontal |
| MDK4LI0000V | 4 | Vertical |
| MDK4LI0000H | 4 | Horizontal |

Figure A-1 shows an overview of an Atlas Developer Kit assembly. The particular assembly shown is for a four axis vertical DK, but the overall elements are similar for one axis developer kits. Horizontal developer kits are also similar except that there is no vertical plate included. Note that the Atlas units shown in the figure are not included with the developer kit and must be purchased separately.

Atlas Developer Kits consists of the mounting and connection hardware only. To create a complete functioning setup one or more Atlas Amplifier units must be ordered separately and then installed onto the DK hardware.

# A.3 Installation and Getting Started

In these instructions it is assumed that you have purchased one of the Magellan Developer Kits, which come with the Pro-Motion exerciser and tuning software. If you have not purchased a Magellan DK then you will still find these instructions useful, however you will use the detailed connections detailed in Section A.4, "Atlas DK Board Reference Information," to connect your system and begin operation.

## A.3.1 Developer Kit Assembly

The first step in getting started with your Atlas Developer Kit is mechanically assembling the hardware that comes with the DK to the Atlas units ordered. This will be described in detail in the next several sections.

It is good practice to wear a grounding strap while handling both the machine controller board and the Atlas units. In addition it is recommended that assembly be undertaken on a surface that dissipates electrostatic charge.

# A.3.2 Vertical Atlas Developer Kit Assembly

## A.3.2.1 Thermal Pad Attachment

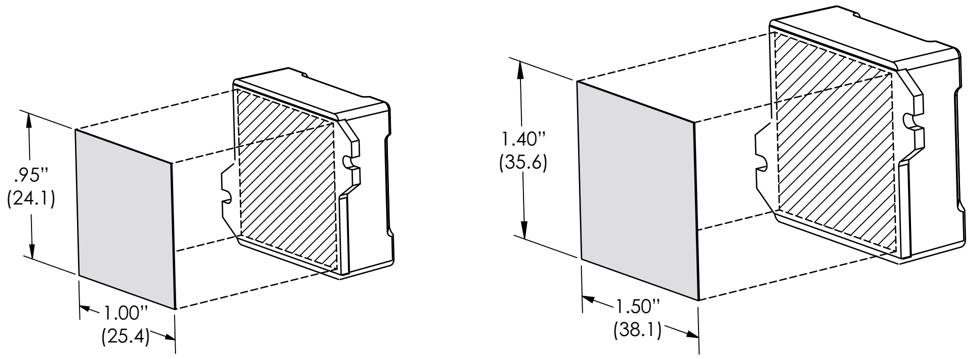



**Figure A-2: Thermal Transfer Material Attachment**

It is very important to have good thermal contact between the Atlas units and the L-bracket heat sink. Therefore the first step will be to attach thermal pads to each vertical Atlas unit to be installed. Horizontal Atlas units do not use a vertical plate for heat sinking and therefore are described in a different section Section A.3.3, "Horizontal Atlas Developer Kit Assembly."

Locate the thermal pads of matching size. Two sets of thermal pads are included in your developer kit reflecting the two available Atlas package sizes - compact or ultra-compact. As shown in the figure above compact units use the larger thermal pad and ultra compact units use the smaller thermal pad.

Next, carefully remove the thin plastic protective sheets on either side of each pre-cut thermal pad and mount onto the Atlas unit, carefully aligning the pads with the Atlas' metallic backing, and applying finger pressure to adhere the pads to the metal. Note that the dimensions of each pad are not exactly square, so it is best to align the pads in the orientation shown in the diagram. Once pressed in place the pads should stay in place, but if required the pads can be removed and remounted.

## A.3.2.2 Installing Atlas Units into the Board

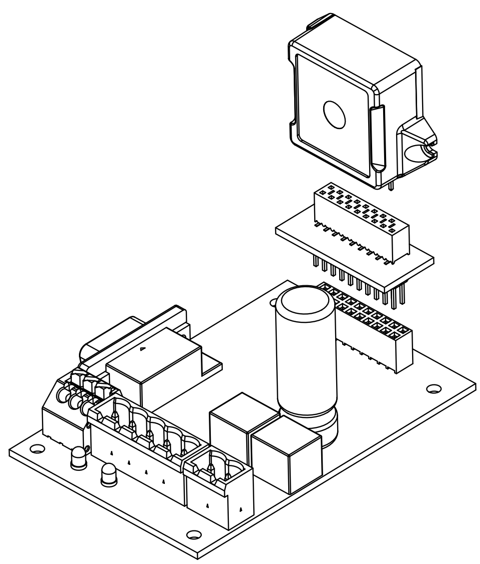


**Figure A-3: Vertical Atlas Installation into DK Board**

To install vertical Atlas units into the Atlas DK board sockets, confirm that the Atlas is oriented correctly, with the metal heat sink surface facing toward the vertical L-bracket plate. Carefully align the Atlas pins to the socket and press firmly down until the Atlas is fully seated in the socket.

With the four axis DK board, if using Atlas units for specific motor types, the motor type of the Atlas should conform to the motor type that will be utilized for that axis. For example if your system has a DC Brush motor at axis #1, and a step motor connected at axis #2, you should install a DC Brush motor Atlas in the axis #1 socket, and a step motor Atlas in the axis #2 socket.

> Extreme care should be taken when installing the Atlas into its socket. Failure to orient the Atlas correctly, or mis-alignment of pins may result in damage to the Atlas units.

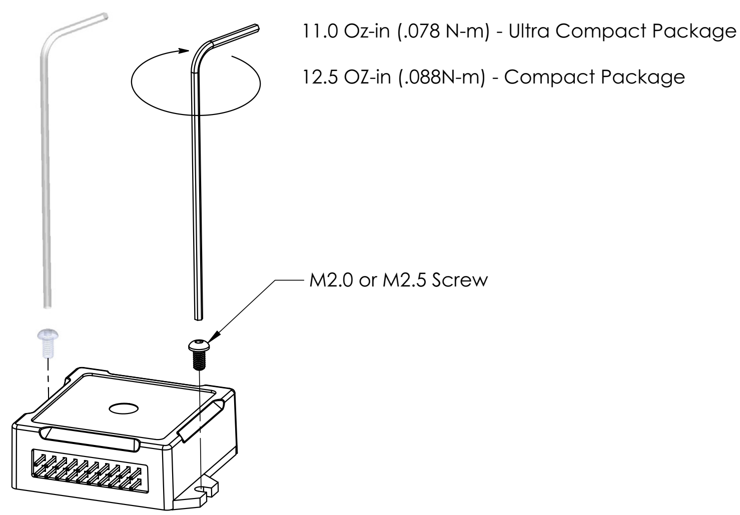
Note that compact Atlas units which are the larger of the two sizes, plug directly into the DK board and do not require the converter card shown in Figure A-3. The smaller ultra compact units, which are the package sizes for the low and medium power Atlas units, require the installation of a conversion card before installation into the Atlas DK.

For any ultra compact Atlas units to be installed onto the board, first mate each ultra-compact Atlas unit to a converter card. Before connecting the Atlas to the converter card, care should be taken to insure that they are oriented correctly, and that all pins align correctly without overhang. Once the ultra-compact Atlas has been properly mated to the converter card, the converter/Atlas assembly can then be inserted into the DK board.

Note that for the four axis Atlas DK board the socket installation location of the compact and ultra compact Atlas units is interchangeable. There is no restriction on the location of compact Atlas units versus the location of ultra compact Atlas units.

### A.3.2.3 Attaching Atlas Units to the Vertical Plate

**Figure A-4: Attaching Atlas Units to Vertical Plate**




Finally, vertical Atlas units should be fastened to the vertical plate. Two screws are used to attach each Atlas and Figure A-4 shows how the screws connect to the vertical plate. For compact Atlas units (high power) the M2.5 screws are used, and for ultra compact Atlas units (low and medium power) M2 screws are used.

Note that the mounting tap hole locations in the vertical plate are different for the compact and ultra compact Atlas units. Use only modest force in attached Atlas units to the vertical plate. Figure A-4 shows this, also providing the torque limit specification for both Atlas types.

Congratulations! You have now completed mechanical assembly of the L-brackets to the Atlas DK board and Atlas units.

## A.3.3 Horizontal Atlas Developer Kit Assembly

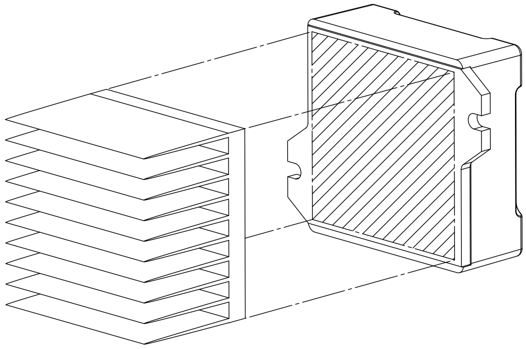


**Figure A-5: Horizontal Atlas Units and Heat Sink**

Horizontal Atlas units and their corresponding Atlas DK boards do not utilize a vertical plate for heat sinking. Rather, as shown in Figure A-5, they use a special finned heat sink included with the DK materials. Before installing the horizontal Atlas units into the Atlas DK board these heat sinks must be attached to each Atlas unit.

To accomplish this, locate the heat sinks of matching size. Two sets of heat sinks are included in your developer kit reflecting the two available Atlas package sizes - compact or ultra-compact. Next, carefully remove the thin plastic protective sheet on the flat side of the heat sink to be attached. Finally, mount the heat sink onto the Atlas unit, carefully aligning the adhesive surface with the Atlas unit's metallic backing, and applying finger pressure to adhere.

Note that the Atlas metal plate dimensions are slightly larger than the dimensions of the heat sink. The exact location of the heat sink relative to the Atlas metal plate is not critical, but it is best to center the heat sink on the Atlas metal plate as much as possible.

### A.3.3.1 Installing Atlas Units into the Board

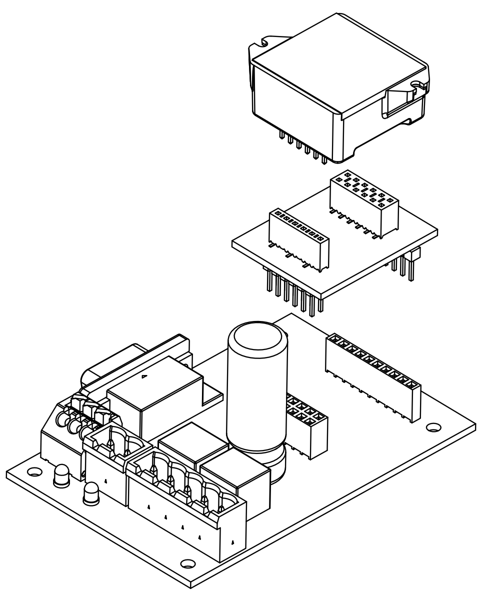


**Figure A-6: Horizontal Atlas Installation into DK Board**

As shown in Figure A-6 horizontal Atlas units are inserted into the horizontal DK boards. The footprint of the horizontal units is quite different than the vertical Atlas units, but otherwise the procedure is similar and is described in Section A.3.2.2, "Installing Atlas Units into the Board."

Similarly for ultra compact horizontal Atlas units, you must install the converter card before installing the Atlas unit(s) into the DK board, however in this case the horizontal converted card, included with horizontal Atlas DKs, is used rather than the vertical converter card.

Once the Atlas units are installed into the DK board assembly of the horizontal DK is complete.

## A.3.4 SPI Bus Connection

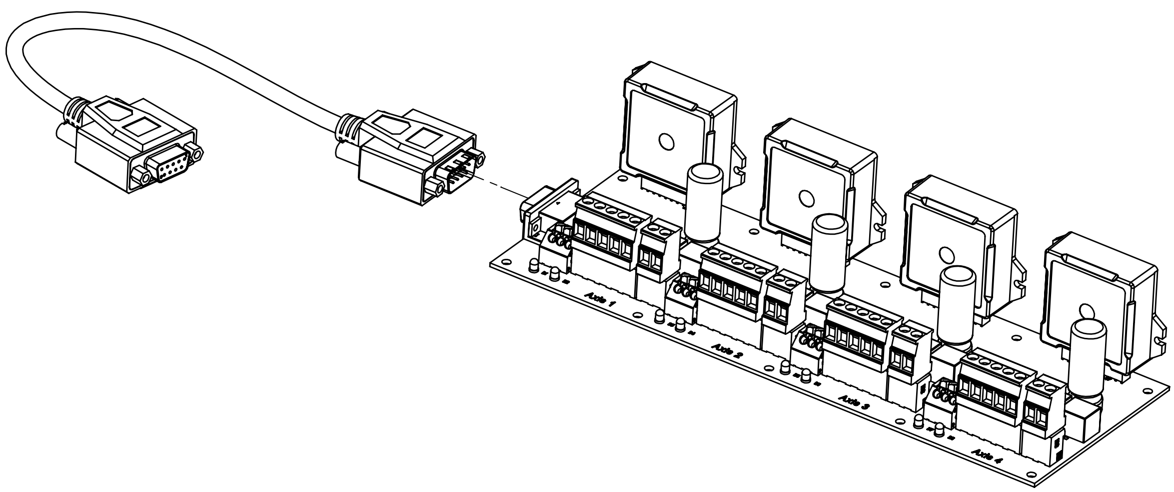
Now that your developer kit is assembled you are ready to connect it to your PC and motion system hardware.

You should plug in the provided 12" DB9 cable at the DB9 connection of the DK board. Once you have plugged in the DB9 cable, you can skip forward to Section A.3.5, "Motor Connections," and continue from there.

> The DB9 connections used with the Atlas DK are not compatible with standard RS232 serial ports. Do not attempt to plug this connector directly into your PC.

> The SPI bus is not designed to operate external modules by cable connection, and therefore in production applications it is recommended that Atlas units be located on the same printed circuit card. Regardless of where Atlas is located, it is the responsibility of the user to ensure that SPI signals are noise free and within Atlas unit's timing specifications.

**Figure A-7: Connecting DB9 Cable to DK Board**



## A.3.5 Motor Connections

Refer to Figure A-8 for detailed information on connector placement. For each Atlas, connect the motor using the chart below and the correct axis-specific 6-terminal screw plug on the DK board, either J2, J5, J8, or J11 for axis 1, 2, 3, or 4 respectively. Use copper wire gauge 14AWG or larger to ensure that all current output requirements can be met.

If you are using a one axis Atlas DK for motor connections and all other connections described in subsequent sections, refer to the description for axis numbers.

| Motor Type | Use Motor Connections | Terminal Screw Plug Labels |
|---|---|---|
| Brushless DC | Motor A, Motor B, Motor C | Mtr A, Mtr B, Mtr C |
| DC Brush | Motor A, Motor B | Mtr A, Mtr B |

| | | |
|---|---|---|
| Step Motor | phase A: Motor A, Motor B<br>phase B: Motor C, Motor D | Mtr A, Mtr B, Mtr C, Mtr D |

## A.3.6 Power Connections

For each Atlas, connect the bus supply voltage (HV) and the associated return ground signal at the correct axis-specific terminal screw plug, either J1, J4, J7, or J10 for axis 1, 2, 3, or 4 respectively. Once again, utilize AWG 14 or larger to ensure that full current demand can be met while operating the unit. The power signals are labeled +HV and GND.

For most installations you will use a single, common power supply to power all Atlas units. However this is not required. If desired, you can operate different Atlas units at different voltages by connecting to different DC supplies.

> While connecting power signals make sure that the power supply is off.

## A.3.7 Enable Signal Connection

You must provide an 'active' enable signal to allow Atlas to operate. There are a few options to accomplish this depending on how you plan to operate your system. A simple approach is to use a short piece of AWG 20 or larger wire to connect the GND connection on the spring clamp Phoenix connector to the Enable input (labeled ~Enab), either J3, J6, J9, or J12 for axis 1, 2, 3, or 4 respectively.

For safety reasons, you may prefer to wire the enable input into a separate switch or E-stop button. Regardless of how it is accomplished, the enable signal must be driven active (low) for Atlas to operate.

## A.3.8 Installing and Connecting to the Magellan DK Card

To set up and install the Magellan DK card refer to the Magellan Developer Kit product that you are using. This manual will help you select jumper settings and make connections to the motor's encoders and other connections. Connect the Atlas DK's SPI bus cable to the appropriate Magellan DK card connector.

Once all connections have been made you should power up the PC (but not the Atlas units) and follow the manual's direction for installing Pro-Motion software. You can run Pro-Motion, check for encoder feedback, etc…. but for axes that utilize Atlas amplifiers, motor output will not yet be operational.

## A.3.9 Powering Up the Atlas Units

Once all connections are made and Pro-Motion is installed and running you are ready to provide power to the Atlas units.

Upon doing so verify that there is no motor movement, all power LEDs are lit, and none of the fault out LED indicators are lit. If any of these conditions are not true, power the Atlas units down and recheck connections.

Once a normal power-up is achieved the Atlas units are ready for operation. You may now use Pro-Motion's Axis Wizard to install and operate your motors, or perform direct manual operations using Pro-Motion's various control menus.

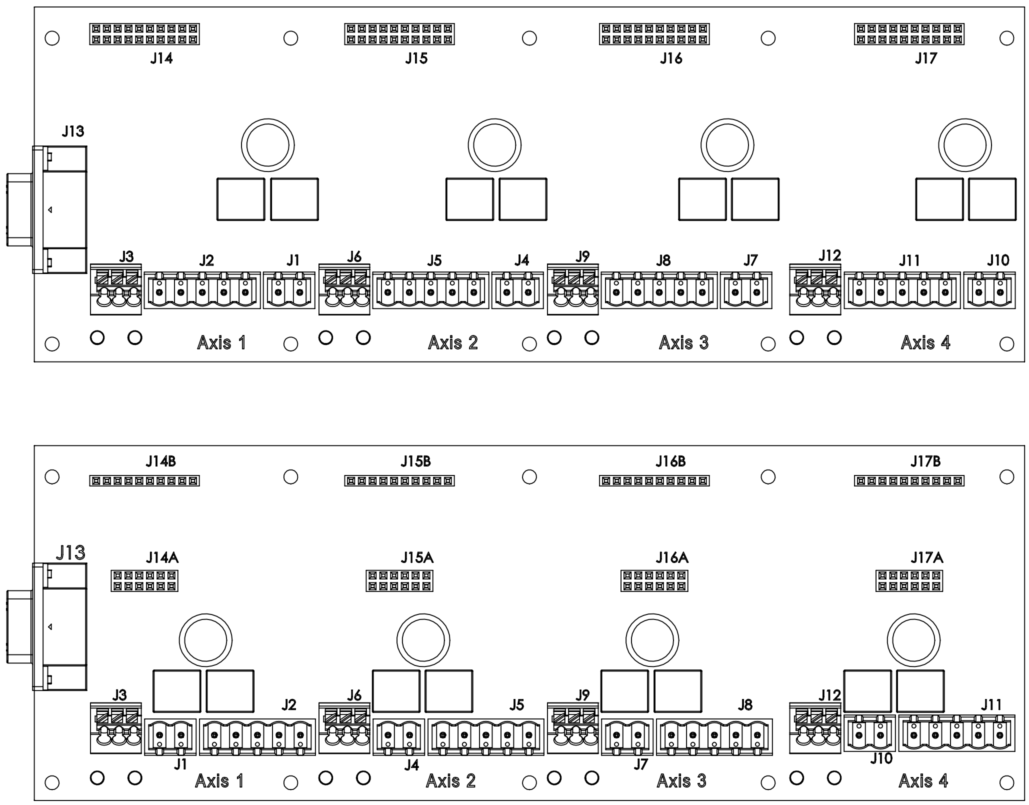
Congratulations! You have successfully installed the Atlas DK.

# A.4 Atlas DK Board Reference Information

The following sections provides detailed information on the electrical characteristics of the Atlas DK boards.

There are four different designs of DK board, representing vertical and horizontal Atlas mount options in both a one axis and four axis configuration.

**Figure A-8: Component Placement of Vertical and Horizontal DK Boards (four-axis version shown)**




The following descriptions apply for the 4-axis DK boards, however the one-axis are similar, only missing axes 2-4.

## A.4.1 J2, J5, J8, and J11 Motor Connectors

J2, J5, J8, and J11 provide terminal screw-style connections to the Atlas motor signals.

| J2, J5, J8 or J11 Connector | | |
|---|---|---|
| **DK Board Label** | **Name** | **Description** |
| Mtr D | Motor D | D Motor connection |
| Mtr C | Motor C | C Motor connection |
| Mtr B | Motor B | B Motor connection |
| Mtr A | Motor A | A Motor connection. |
| GND | Mtr_Gnd | Ground return for Motor and HV |

## A.4.2 J1, J4, J7, J10 Power Connectors

J1, J4, J7, and J10 provide terminal screw-style connections to supply the Atlas power signals.

| J1, J4, J7 and J10 Connectors | | |
|---|---|---|
| **DK Board Label** | **Name** | **Description** |
| HV | HV | Motor Supply Voltage |
| Pwr_Gnd | Pwr_Gnd | Ground return for Motor Supply |

## A.4.3 J3, J6, J9, J12 Signal Connectors

J3, J6, J9, and J12 provide terminal screw-style connections to supply the Atlas signal connections.

| J3, J6, J9 and J12 Connectors | | |
|---|---|---|
| **DK Board Label** | **Name** | **Description** |
| ~Enab | Enable | Enable input |
| Flt | FaultOut | FaultOut output |
| GND | GND | Ground return for *Enable* and *FaultOut* signals |

## A.4.4 Quick Connect Motor Type Chart

| Motor Type | Connections |
|---|---|
| Brushless DC | Motor A, Motor B, Motor C |
| DC Brush | Motor A, Motor B |
| Step Motor | phase A: Motor A, Motor B<br>phase B: Motor C, Motor D |

# A.4.5 J13 DB9 Connector

## A.4.5.1 SPI Communications

J13 is used to provide SPI communications between the Atlas DK card and a Magellan DK card or the user's motion control system.

Here are the pinouts for J13 when used for SPI communications

| J13 Connector | | |
|---|---|---|
| **Pin** | **Name** | **Description** |
| 1 | ~SPICS3 | SPI chip select for Atlas #3 |
| 2 | ~SPICS2 | SPI chip select for Atlas #2 |
| 3 | Shield | Cable shield connection |
| 4 | GND | Ground |
| 5 | SPISO | SPI Slave Out |
| 6 | ~SPICS1 | SPI chip select for Atlas #1 |
| 7 | ~SPICS4 | SPI chip select for Atlas #4 |
| 8 | SPIClk | SPI Clock |
| 9 | SPISI | SPI Slave In |

## A.4.5.2 Pulse & Direction Mode

J13 can also be used to provide pulse & direction signals to a single Atlas.

Here are the pinouts for J13 when used in pulse & direction signal mode

| J13 Connector | | |
|---|---|---|
| **Pin** | **Name** | **Description** |
| 1 | not used | |
| 2 | not used | |
| 3 | Shield | Cable shield connection |
| 4 | GND | Ground |
| 5 | not used | |
| 6 | AtRest | Pulse & direction mode *AtRest* signal |
| 7 | not used | |
| 8 | Pulse | Pulse & direction mode *Pulse* signal |
| 9 | Direction | Pulse & direction mode *Direction* signal |

# A.4.6 Atlas Connections

The DK board connects to the Atlas units via sockets at J14, J15, J16, and J17. The tables below show the Atlas connections for these connectors

## A.4.6.1 Vertical Unit Connections

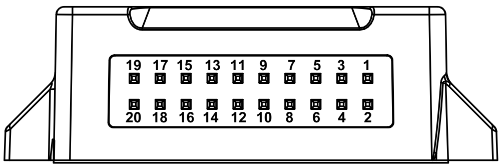



Figure A-9: Vertical Unit Pinouts

**J14, J15, J16, & J17 Connectors**

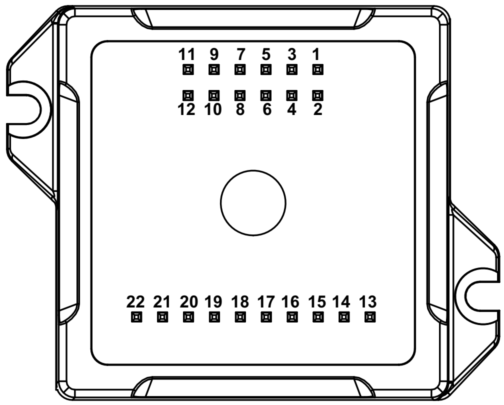
| Pin | Name | Pin | Name |
|---|---|---|---|
| 1 | Pwr_Gnd | 2 | Pwr_Gnd |
| 3 | HV | 4 | HV |
| 5 | Motor A | 6 | Motor A |
| 7 | Motor B | 8 | Motor B |
| 9 | Motor C | 10 | Motor C |
| 11 | Motor D | 12 | Motor D |
| 13 | ~Enable | 14 | FaultOut |
| 15 | 5V | 16 | GND |
| 17 | ~SPICS/AtRest | 18 | SPISI/Direction |
| 19 | SPIClk/Pulse | 20 | SPISO |

Refer to Figure A-8 for connector locations.

The pins are 0.1 inch spacing and 0.025 inch pin width.

### A.4.6.2 Horizontal Unit Connections

**Figure A-10: Horizontal Unit Pinouts**




**J14A, J15A, J16A, & J17A Connectors**

| Pin | Name | Pin | Name |
|---|---|---|---|
| 1 | Motor D | 2 | Motor D |
| 3 | Motor C | 4 | Motor C |
| 5 | Motor B | 6 | Motor B |
| 7 | Motor A | 8 | Motor A |
| 9 | HV | 10 | HV |
| 11 | Pwr_Gnd | 12 | Pwr_Gnd |

**J14B, J15B, J16B, & J17B Connectors**

| | | | |
|---|---|---|---|
| 13 | 5V | 14 | GND |
| 15 | ~Enable | 16 | FaultOut |
| 17 | GND | 18 | ~SPICS/AtRest |
| 19 | SPISO | 20 | SPISI/Direction |
| 21 | SPIClk/Pulse | 22 | GND |

The pins are 0.1 inch spacing and 0.025 inch pin width.

## A.4.7 Compact To Ultra Compact Package Signal Converters

When ultra compact package Atlas units are installed in the Atlas DK, signal converter cards are installed between the DK boards and the Atlas unit. Two converter formats are provided, one for horizontal Atlas units and one for Vertical units. These devices convert signals from the DK card's compact package format to the ultra compact format. These converters may also be used in other products designed for the compact package format such as the Prodigy/CME Machine Controller cards.

### A.4.7.1 Vertical Ultra Compact Converter Pinouts

The following section shows the connections provided by the vertical converter.

| Compact Package Pin | Ultra Compact Package Pin | Name |
|---|---|---|
| 1, 2 | 4 | Pwr_Gnd |
| 3, 4 | 1 | HV |
| 5, 6 | 3 | Motor A |
| 7, 8 | 6 | Motor B |
| 9, 10 | 5 | Motor C |
| 11, 12 | 7 | Motor D |
| 13 | 11 | ~Enable |
| 14 | 12 | FaultOut |
| 15 | 14 | 5V |
| 16 | 13 | GND |
| 17 | 16 | ~SPICS/AtRest |
| 18 | 18 | SPISI/Direction |
| 19 | 17 | SPIClk/Pulse |
| 20 | 15 | SPISO |

### A.4.7.2 Horizontal Ultra Compact Converter Pinouts

The following section shows the connections provided by the horizontal converter.

| Compact Package Pin | Ultra Compact Package Pin | Name |
|---|---|---|
| 1, 2 | 1 | Motor D |
| 3, 4 | 2 | Motor C |
| 5, 6 | 3 | Motor B |
| 7, 8 | 4 | Motor A |
| 9, 10 | 5 | HV |
| 11, 12 | 6 | Pwr_Gnd |
| 13 | 11 | 5V |
| 14 | 12, 15 | GND |
| 15 | 14 | ~Enable |
| 16 | 13 | FaultOut |
| 17 | 12, 15 | GND |
| 18 | 10 | ~SPICS/AtRest |
| 19 | 9 | SPISO |
| 20 | 7 | SPISI/Direction |
| 21 | 8 | SPIClk/Pulse |
| 22 | 12, 15 | GND |

## A.4.8 LED Indicators

The Atlas DK board has two LEDs. The green LED, when lit, indicates that Atlas is receiving valid power input power at HV. The red LED, when lit, indicates that an Atlas FaultOut condition is active.

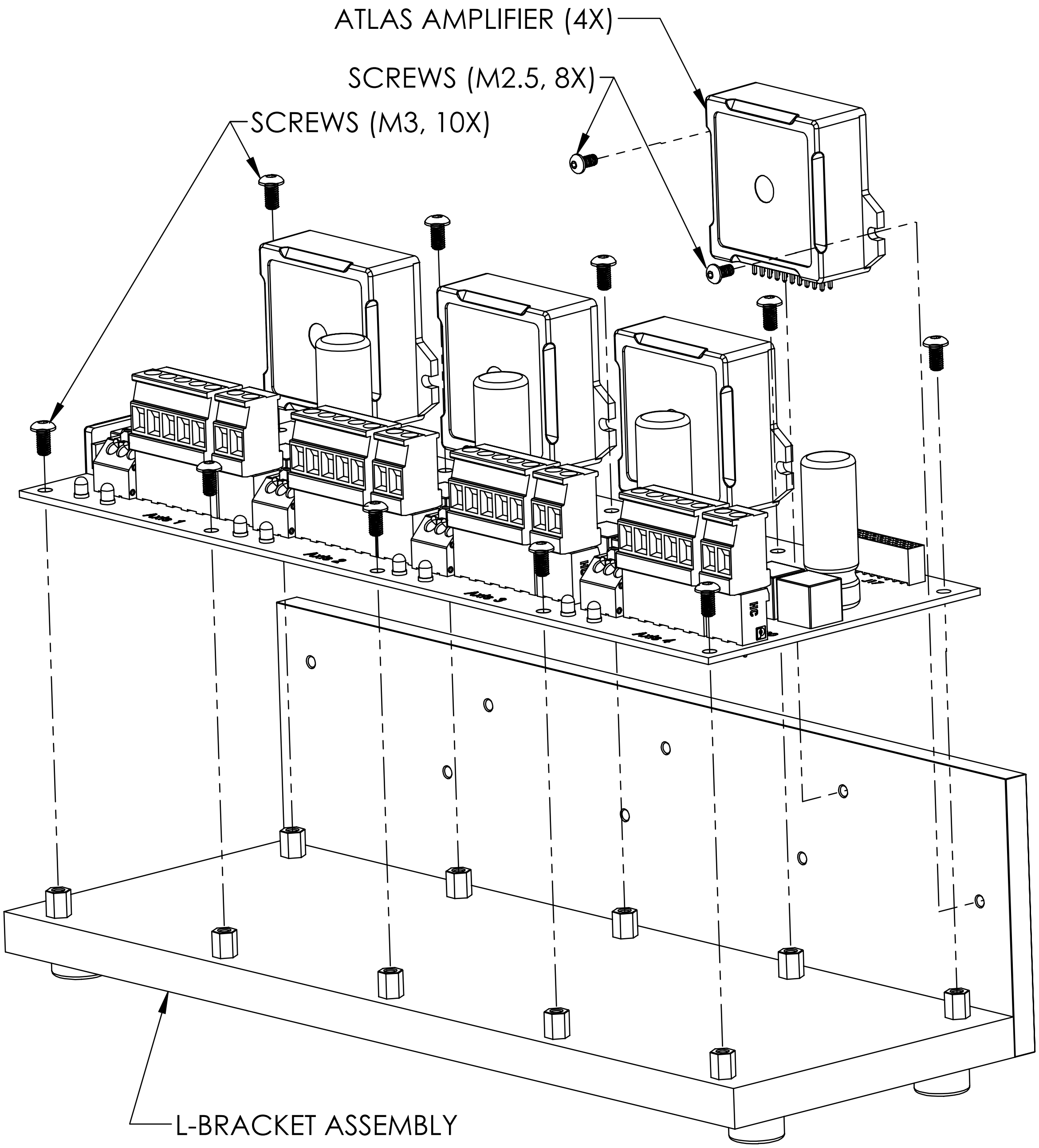
# A.5 L-Bracket

The Atlas DK comes with mounting plates that provide extra mechanical stability and heat sinking during prototyping. Depending on the type of Atlas that you are using, you will use just the base plate or the base plate and vertical plate in the “L” configuration.

L-bracket hardware is provided in the one axis configuration and four axis configuration.

Normally, the DK boards are fully assembled into the base plate. If for whatever reason you need to disassemble or assemble the base plate to the DK board however, you can assemble these components yourself. To assist with this, an assembly drawing is shown in Figure A-11. In addition, you will need 1.5 mm and 2 mm hex wrenches to assemble the DK board, Atlas units, and L-bracket together.

**Figure A-11: Mounting Atlas to L-bracket Plates (four-axis, vertical version shown)**




## A.5.1 Mounting L-bracket to Other Hardware

To maximize heat sinking capacity you may choose to mount the vertical L-bracket piece to your own hardware. For best thermal performance, a material such as Sil-Pad thermal grease or phase change material should be utilized between metal interfacing layers.

To connect to the vertical plate use four (4) M4 screws threaded into the provided threaded holes in the vertical plate or use four (4) M3 screws with nuts and washers to fasten through from the front.

The diagrams below show the location of these mounting holes for both the 4-axis and 1-axis vertical plates.

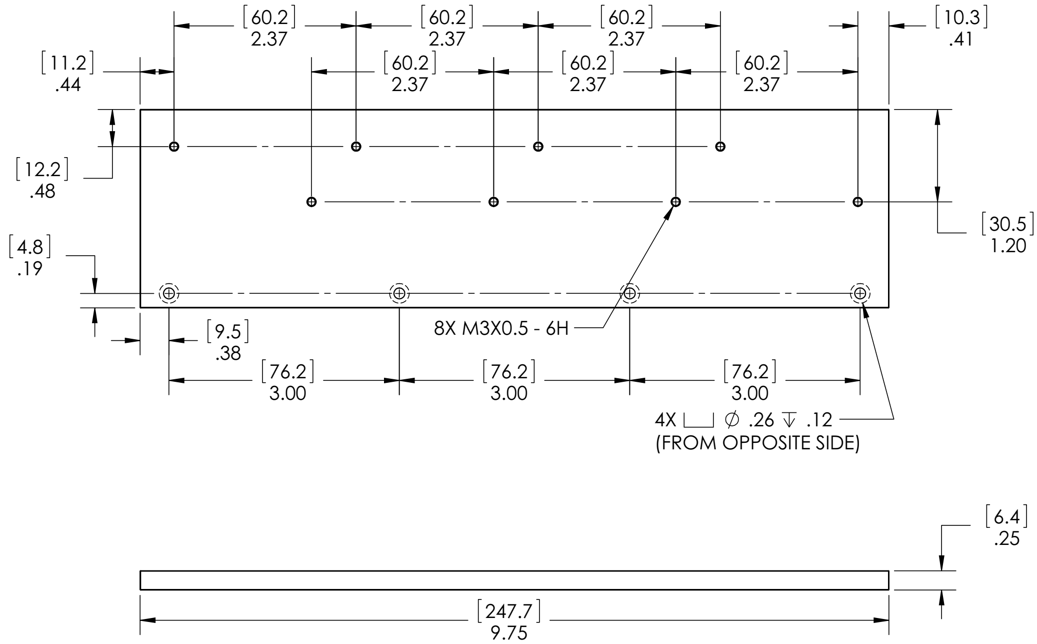



**Figure A-12: Top and Front Views of Four-Axis Horizontal Atlas DK L-bracket Vertical Plate**

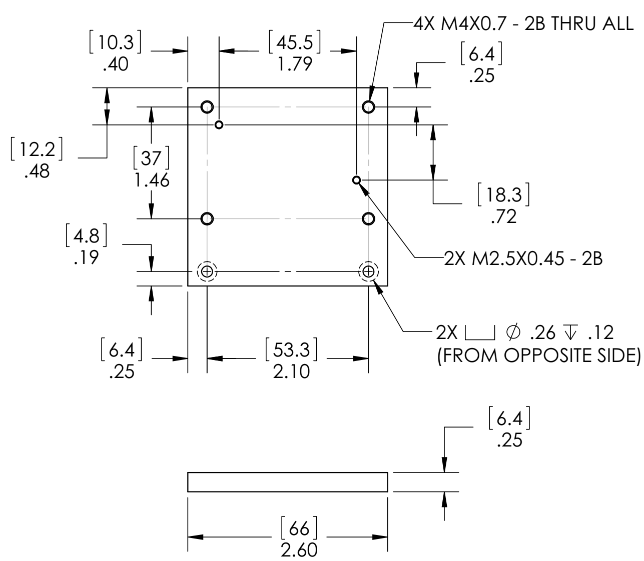



**Figure A-13: Top and Front Views of One-Axis Horizontal Atlas DK L-bracket Vertical Plate**

*This page intentionally left blank.*

# B. Application Notes

B

### *In This Appendix*

- General Design Notes
- Brushless DC Atlas With Single-Axis MC58113 Motion Control IC
- DC Brush & Step Motor Atlas With Multi-Axis Magellan
- Step Motor Atlas Operating In Pulse & Direction Mode
- DC Brush Atlas With PIC Microcontroller
- Step Motor Atlas With ARM Microcontroller
- Atlas Interfacing Via a Daughter Card
- Multi-Motor Atlas With Single-Axis MC58113 Motion Control IC

## B.1 General Design Notes

This section provides general information intended to be useful for the design and development of PCBs which include Atlas® Digital Amplifiers.

**Pins**

The recommended PCB drill hole is 0.040" for all Atlas pins and the pin pitch (spacing) is 0.100".

Refer to Section 2.2, "Physical Dimensions" for pin dimensions and other information on mechanical dimensions of the Atlas units.

**PCB Layers & Traces**

Use of a minimum of four layers is recommended for a number of reasons, especially because it allows greater use of ground planes. Ground planes on one or more PCB layers reduce EMI and minimize cross talk between the high current, high voltage portions of the PCB (the traces connected to the power pins HV, Motor A-D, Pwr_GND) and the lower voltage, more noise sensitive portions of the PCB (signal traces connected ). However for most PCBs that also have active circuitry on the board, particularly high density parts such as microcontrollers, FPGAs, etc... six, eight, or even more layers is common.

In addition to ground planes, especially for the Power Connector signals, consideration should be given to the trace dimensions. Current carrying capacity of traces and associated issues such as thermal rise in the trace are determined by a combination of the PCB copper thickness, the trace width, and the trace length. Although beyond the scope of this document it is possible to calculate the thermal rise and increase in resistance that occurs in each trace based on the dimensions and current flow through it.

For a complete PCB design example PMD's Atlas DK interconnect board layout is available. This eight-layer board demonstrates use of ground planes, PCB layer thickness, and trace dimensions. Contact PMD support for PMD's Atlas DK interconnect board layout file.

**Power Input**

Atlas is powered through pin pairs HV and Pwr_Gnd, and the power source is a transformer-isolated DC power supply. When unregulated DC power supply is used the output voltage with respect to its output power/current should meet the full Atlas operating range specification. Be aware that for motors operating with significant inertia, during deceleration, Atlas may convert excess kinetic energy to electrical energy and feed the energy back to the DC power supply input. The power supply therefore should be able to absorb or "dump" this regenerated energy so that the increased bus voltage will not trigger an Atlas over-voltage event, or otherwise damage the power supply or other attached devices. An input capacitor can be used to absorb the regenerated energy with E=C*V*deltaV where E is the kinetic energy, C is the capacitance of the input capacitance, V is the DC bus voltage and deltaV is the allowed voltage increase. For some regulated power sources the regenerated energy may interfere with the power source's operation. If that is the case, you may consider adding a diode between the regulated power source and the input capacitor bank.

The Atlas DC supply voltage range is 12-48V for low and medium power units and 12-56V for high power units. The Atlas amplifier has an internal capacitance of 33µF on the DC bus (HV line).

The Pwr_Gnd and GND pins are connected together inside the Atlas, and at a system level they refer to the same ground. Pwr_Gnd, the current return path for the power train, is paired with HV and may therefore be noisy. GND is the reference for the SPI signals and other digital control signals. These signals require a quiet ground reference. To ensure optimal performance, star grounding is recommended for component placement and layout. That is, Pwr_Gnd and GND should be connected to the system ground very close to Atlas, and the two ground paths should be kept away from each other.

There is a third current return path stemming from the high frequency component of the motor winding current. Atlas drives motor windings with pulse-width modulated (PWM) signals. Although the sum of the average winding currents is zero, the high frequency PWM signal may couple to the ground plane and induce noise into other circuits. Therefore, depending on your application, you may consider utilizing a shielded motor cable to provide a current return path. If utilized, its ground point should be very close to, or the same, as Pwr_Gnd.

# B.2 Brushless DC Atlas With Single-Axis MC58113 Motion Control IC

The following schematic shows a Brushless DC Atlas Amplifier connected to a single-axis Magellan.

## B.2.1 Atlas Power Supply

In the schematic on the facing page the ground pin in the Atlas amplifier pinouts has been given two unique signal names (Pwr_GND) to distinguish it from the signal GND pins. Power_GND and GND should be segregated at a layout level. Pwr_GND is the current return path for the motor power train, and may therefore be noisy. GND is the reference for digital control signals and these signals require a quiet ground reference. Typically the layout should isolate Pwr_GND and GND. Doing so will keep the noise on the power train from the rest of the digital circuits and improve noise immunity. Certain scenarios may benefit from connecting Pwr_GND and GND which is at the discretion of the designer.

The Atlas digital amplifier has an internal capacitance of 33µF on the DC bus (HV line). Depending on the application you may consider adding additional decoupling capacitance. Electrolytic capacitors may be used to increase energy absorption of supply capacity, and if needed ceramic capacitors may be used to lower ESR.

## B.2.2 Atlas SPI Interface

Atlas receives control commands through an SPI interface and functions as an SPI slave. Atlas SPI communication is enabled when ~SPICS is pulled down.

To ensure optimal SPI communication, please consider the following layout recommendations:

1 Keep traces short and use 45 degree corners instead of 90 degree corners.

2 All SPI signal traces should be located next to a continuous ground plane, or if possible, between two continuous ground planes.

3 Keep traces away from other noisy and high speed signal traces. Alternatively, run ground traces along with these signals as a shield.

4 When multiple Atlas modules are used, keep the SPI signal stubs short.

Note that the Atlas Development Kit layout can be used as a layout reference.

### B.2.3 Atlas ~Enable and FaultOut Signals

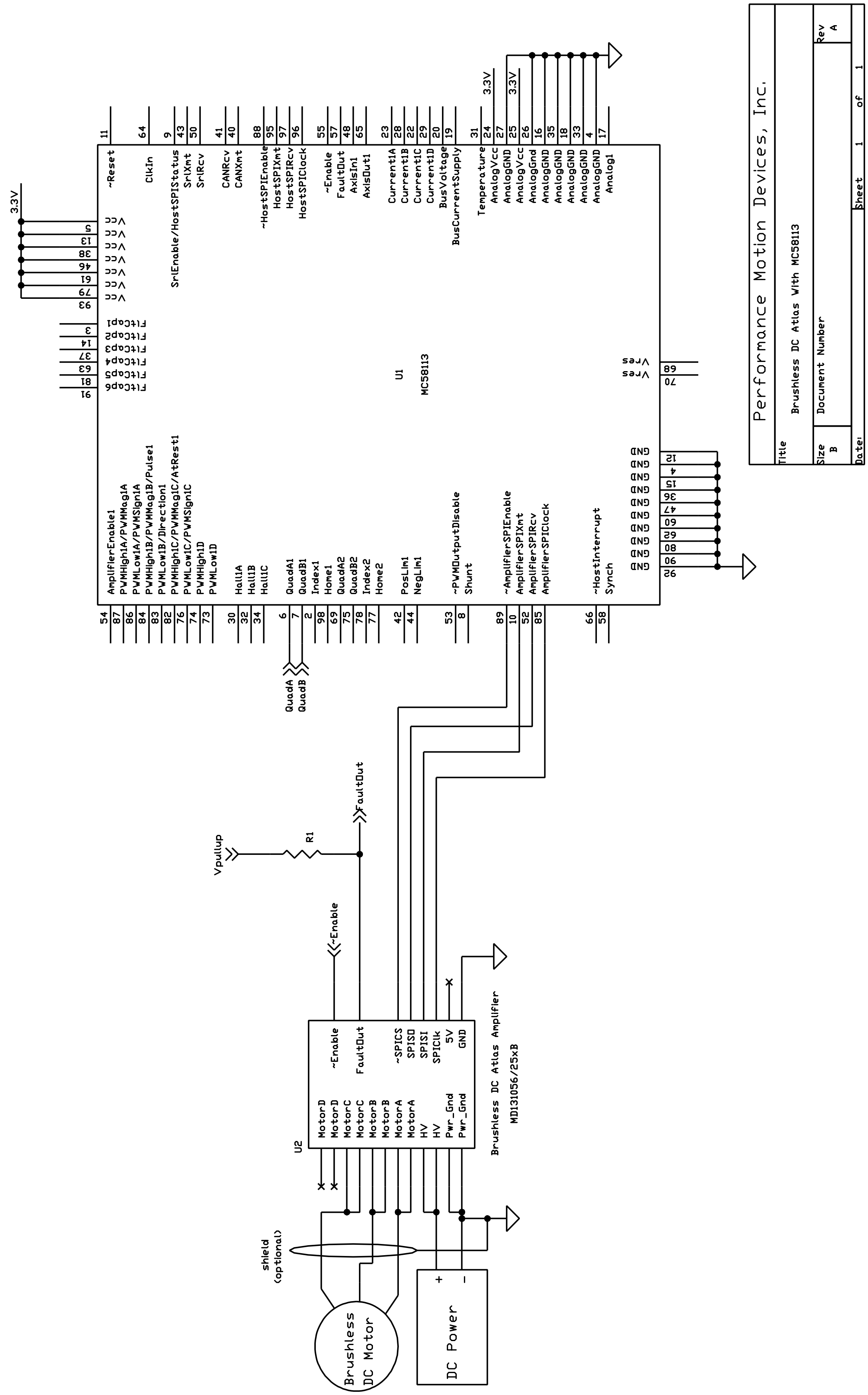
Atlas has one dedicated input signal, ~Enable, which must be pulled low for the Atlas output stage to be active.

FaultOut is a dedicated output. During normal operation it outputs low. When a fault occurs it will go into a high impedance state. In this example, FaultOut is pulled up by Vpullup through resistor R1. Vpullup can be up to 24V to meet the system requirement. For example, if the fault signal is wired to a 5V TTL input, Vpullup can be 5V.

### B.2.4 Magellan MC58113 Configuration

In this schematic the SPI master is a single axis Magellan MC58113. Only the connections with Atlas are shown. For complete MC58113 wiring, please refer to the MC58113 electrical specifications. Depending on the Magellan commutation method selected the feedback signals HallA, HallB, HallC and ~Index are optional.

**Figure B-1: Brushless DC Atlas With Single-Axis Magellan**



# B.3 DC Brush & Step Motor Atlas With Multi-Axis Magellan

The following schematic shows a two-axis application with one DC Brush Atlas Amplifier and one step motor Atlas amplifier controlled by a multi-axis Magellan.

## B.3.1 Atlas Power Input and Motor Output

Atlas is powered through pin pairs HV and Pwr_Gnd, and the power source is a transformer-isolated DC power supply. In this application the two Atlases share the same power supply. Alternatively they could be powered independently so that different motor voltages could be used.

For DC Brush motors pins MotorA and MotorB are wired to motor windings Motor+ and Motor-, respectively. Pins MotorC and MotorD are left un-connected.

For step motors pins MotorA, MotorB, MotorC and MotorD are wired to motor windings A+, A-, B+ and B-, respectively.

Please refer to B.1 for layout and wiring recommendations on power input and motor outputs.

## B.3.2 Atlas SPI Interface

Atlas receives control commands through an SPI interface and functions as an SPI slave. Atlas SPI communication is enabled when ~SPICS is pulled down. Only one Atlas can be enabled at any given time.

Please refer to B.1 for layout recommendation on SPI interface.

## B.3.3 Atlas ~Enable and FaultOut Signals

Atlas has one dedicated input signal, ~Enable, which must be pulled low for the Atlas output stage to be active.

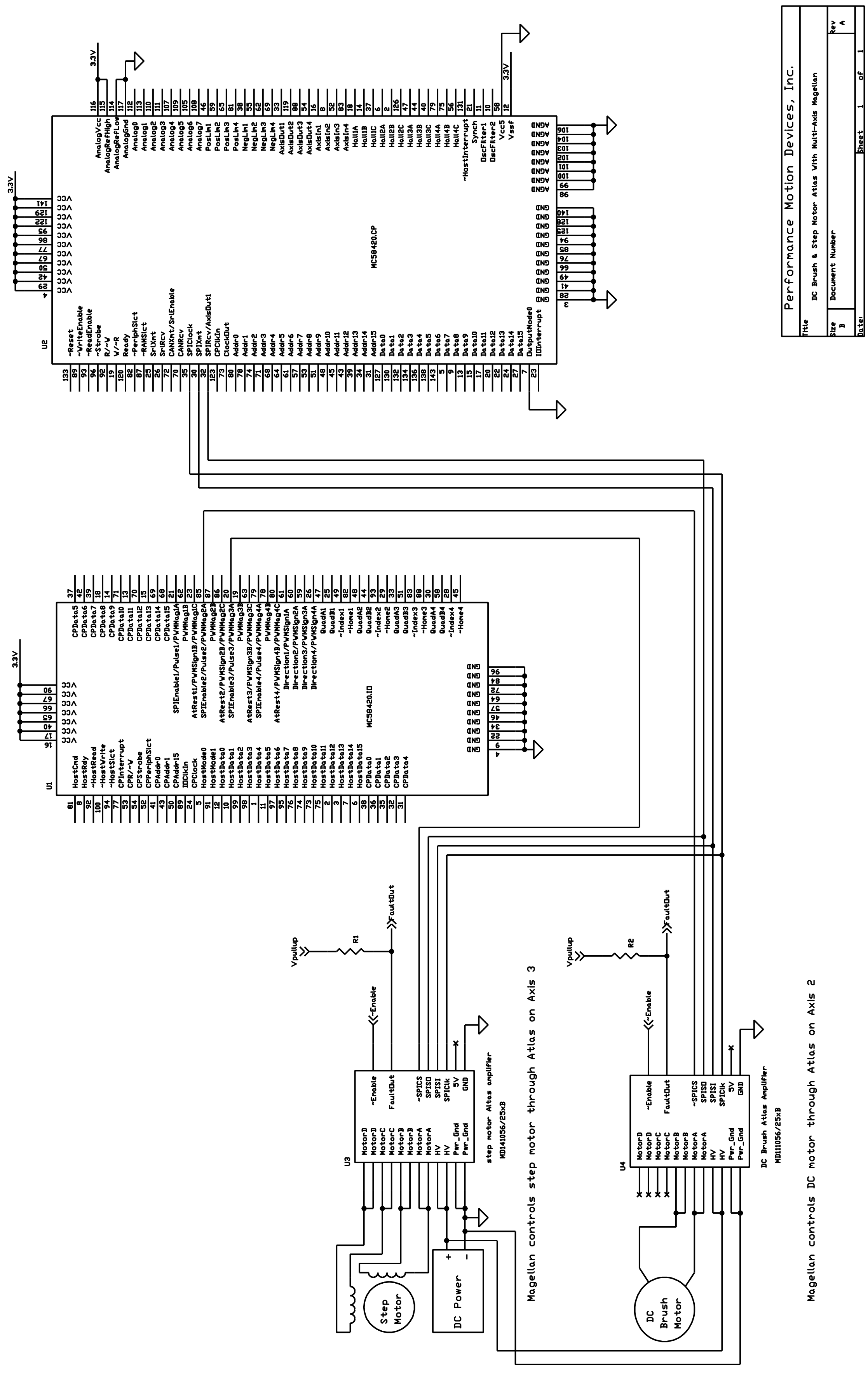
FaultOut is a dedicated output. During normal operation it outputs low. When a fault occurs it will go into a high impedance state. In this example, FaultOut is pulled up by Vpullup through resistor R1. Vpullup can be up to 24V to meet the system requirement. Each Atlas may use a different Vpullup voltage, for example, if the fault signal is wired to a 5V TTL input, Vpullup can be 5V.

## B.3.4 Magellan MC58420 Configuration

In this schematic the SPI master is a four-axis Magellan MC58420. Only the connections with Atlas are shown. For complete Magellan wiring, please refer to the MC58420 electrical specifications.

The MC58420 is configured to default to Atlas motor output by tying pin 7, OutputMode0, to ground. In this example axis 2 and axis 3 are under control. The MC58420 sends torque commands to the DC Brush Atlas by pulling SPIEnable2 low, and sends position commands to the step motor Atlas by pulling SPIEnable3 low.

**Figure B-2: DC Brush & Step Motor Atlas With Multi-Axis Magellan**



# B.4 Step Motor Atlas Operating In Pulse & Direction Mode

The following schematic shows Atlas operated in pulse & direction mode controlled by a single axis Magellan. Note that any source of pulse & direction signals, such as a microprocessor or other dedicated motion control IC, may be substituted for the Magellan in this schematic.

## B.4.1 Atlas Power Input and Motor Outputs

Atlas is powered through pin pairs HV and Pwr_Gnd, and the power source is a transformer-isolated DC power supply.

For step motors pins MotorA, MotorB, MotorC and MotorD are wired to motor windings A+, A-, B+ and B-, respectively.

Please refer to B.1 for layout and wiring recommendations on power input and motor outputs.

## B.4.2 Atlas Pulse & Direction Interface

When in pulse & direction signal mode, Atlas receives pulse, direction and AtRest signals as shown in the schematic. When operated in pulse & direction signal mode SPI communications are not available.

## B.4.3 Atlas ~Enable and FaultOut Signals

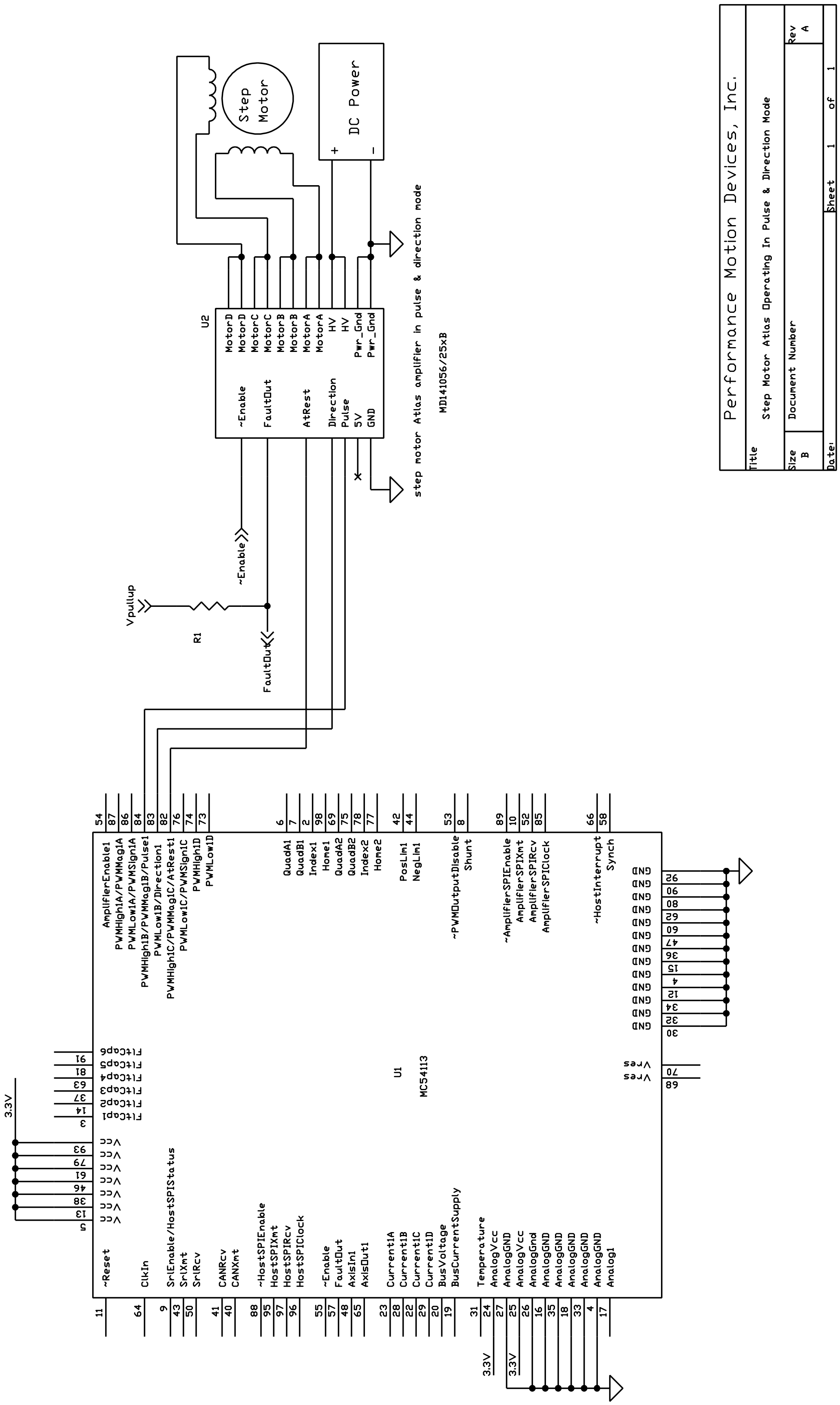
Atlas has one dedicated input signal, ~Enable, which must be pulled low for the Atlas output stage to be active.

FaultOut is a dedicated output. During normal operation it outputs low. When a fault occurs it will go into a high impedance state. In this example, FaultOut is pulled up by Vpullup through resistor R1. Vpullup can be up to 24V to meet the system requirement. For example, if the fault signal is wired to a 5V TTL input, Vpullup can be 5V.

## B.4.4 Magellan MC54113 Configuration

In this schematic the SPI master is a single-axis Magellan MC54113 configured for pulse & direction signal output. Only the connections with Atlas are shown. For complete Magellan wiring, please refer to the MC58113 electrical specifications.

**Figure B-3: Step Motor Atlas Operating In Pulse & Direction Mode**



# B.5 DC Brush Atlas With PIC Microcontroller

The following schematic shows a DC Brush Atlas amplifier connected to a Microchip Technologies' PIC microcontroller. Atlas receives torque commands through the PIC's SPI interface.

A wide variety of microcontrollers, DSP-type devices, or FPGAs supporting SPI interfaces can control Atlas directly. Microchip's dsPIC33FJ64GS606 is used in this example. It supports encoder inputs and other feedback inputs. Users design their own motion control algorithms on the microcontroller, which in turn commands Atlas to drive the motor.

## B.5.1 Atlas Power Input and Motor Output

Atlas is powered through pin pairs HV and Pwr_Gnd, and the power source is a transformer-isolated DC power supply.

For DC Brush motors pins MotorA and MotorB are wired to motor windings Motor+ and Motor-, respectively. Pins MotorC and MotorD are left un-connected.

Please refer to B.1 for layout and wiring recommendation on power input and motor outputs.

## B.5.2 Atlas SPI Interface

Atlas functions as an SPI slave, receiving control commands from the PIC through its SPI interface. Atlas SPI communication is enabled when ~SPICS is pulled down.

Please refer to B.1 for layout recommendation on SPI interface.

## B.5.3 Atlas ~Enable and FaultOut Signals

Atlas has one dedicated input signal, ~Enable, which must be pulled low for the Atlas output stage to be active.

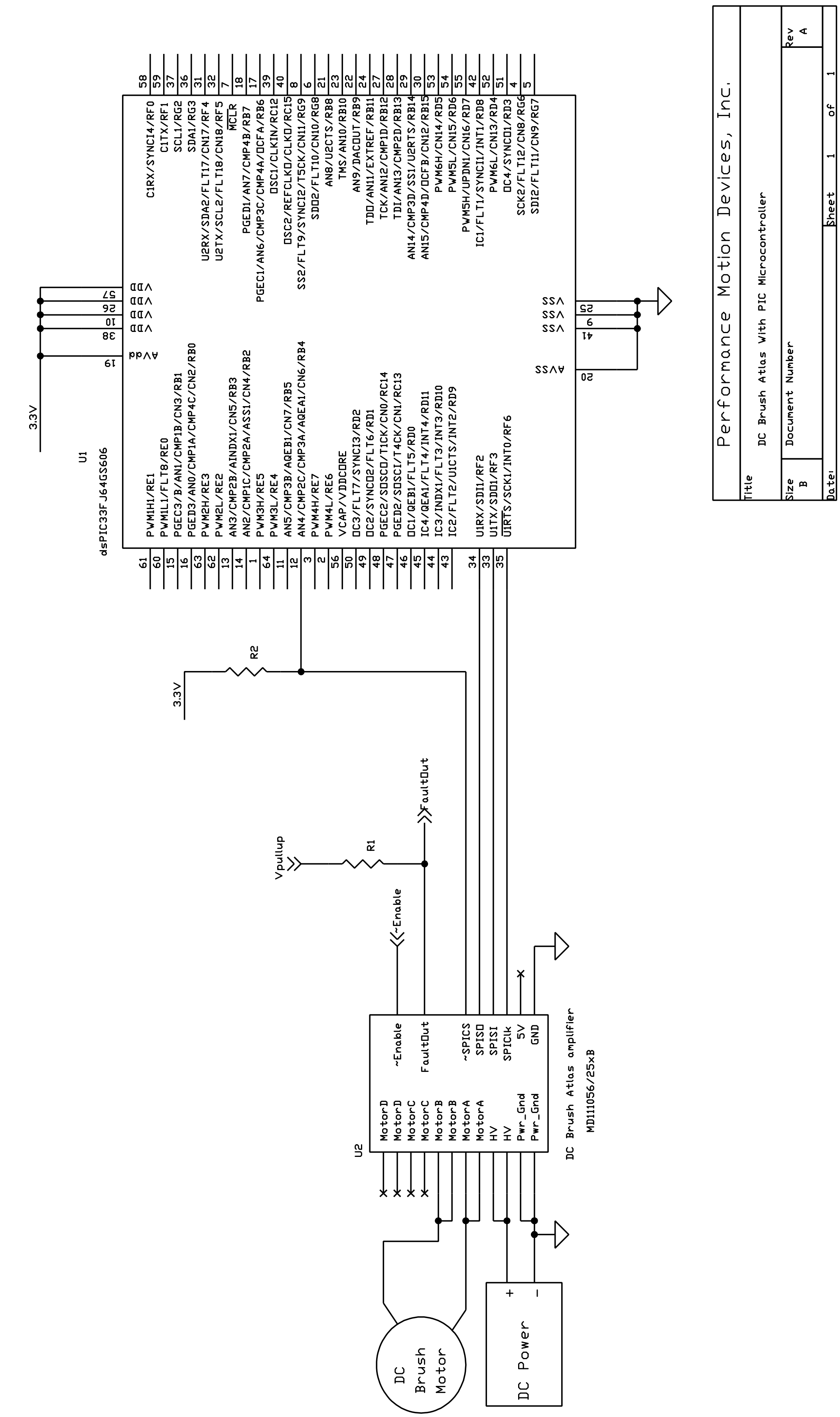
FaultOut is a dedicated output. During normal operation it outputs low. When a fault occurs it will go into a high impedance state. In this example, FaultOut is pulled up by Vpullup through resistor R1. Vpullup can be up to 24V to meet the system requirement. For example, if the fault signal is wired to a 5V TTL input, Vpullup can be 5V.

## B.5.4 Microcontroller Configuration

In this schematic, the host controller is Microchip's dsPIC33FJ64GS606. The microcontroller's SPI port (SDI1/SDO1/SCK1) is used for SPI communication.

In this example output pin #12 of U1 (PIC processor) is used to control Atlas' ~SPICS input. ~SPICS has an internal pull-up, therefore, during power up and after reset, the control pin should be in high-impedance or output high state to disable the SPI. Resistor R2 is optional.

**Figure B-4: DC Brush Atlas With PIC Microcontroller**



# B.6 Step Motor Atlas With ARM Microcontroller

The following schematic shows a step motor Atlas amplifier connected to an STMicroelectronic's ARM microcontroller. Atlas receives torque commands through the ARM's SPI interface.

A wide variety of microcontrollers, DSP-type devices, or FPGAs supporting SPI interfaces can control Atlas directly. STMicroelectronic's STR912FAZ44H6T is used in this example. Users design their own motion control algorithms on the microcontroller, which in turn commands Atlas to drive the motor.

## B.6.1 Atlas Power Input and Motor Outputs

Atlas is powered through pin pairs HV and Pwr_Gnd, and the power source is a transformer-isolated DC power supply.

For step motors pins MotorA, MotorB, MotorC and MotorD are wired to motor windings A+, A-, B+ and B-, respectively.

Please refer to B.1 for layout and wiring recommendations on power input and motor outputs.

## B.6.2 Atlas SPI Interface

Atlas functions as an SPI slave, receiving control commands from the ARM through its SPI interface. Atlas SPI communication is enabled when ~SPICS is pulled down.

Please refer to B.1 for layout recommendation on SPI interface.

## B.6.3 Atlas ~Enable and FaultOut Signals

Atlas has one dedicated input signal, ~Enable, which must be pulled low for the Atlas output stage to be active.

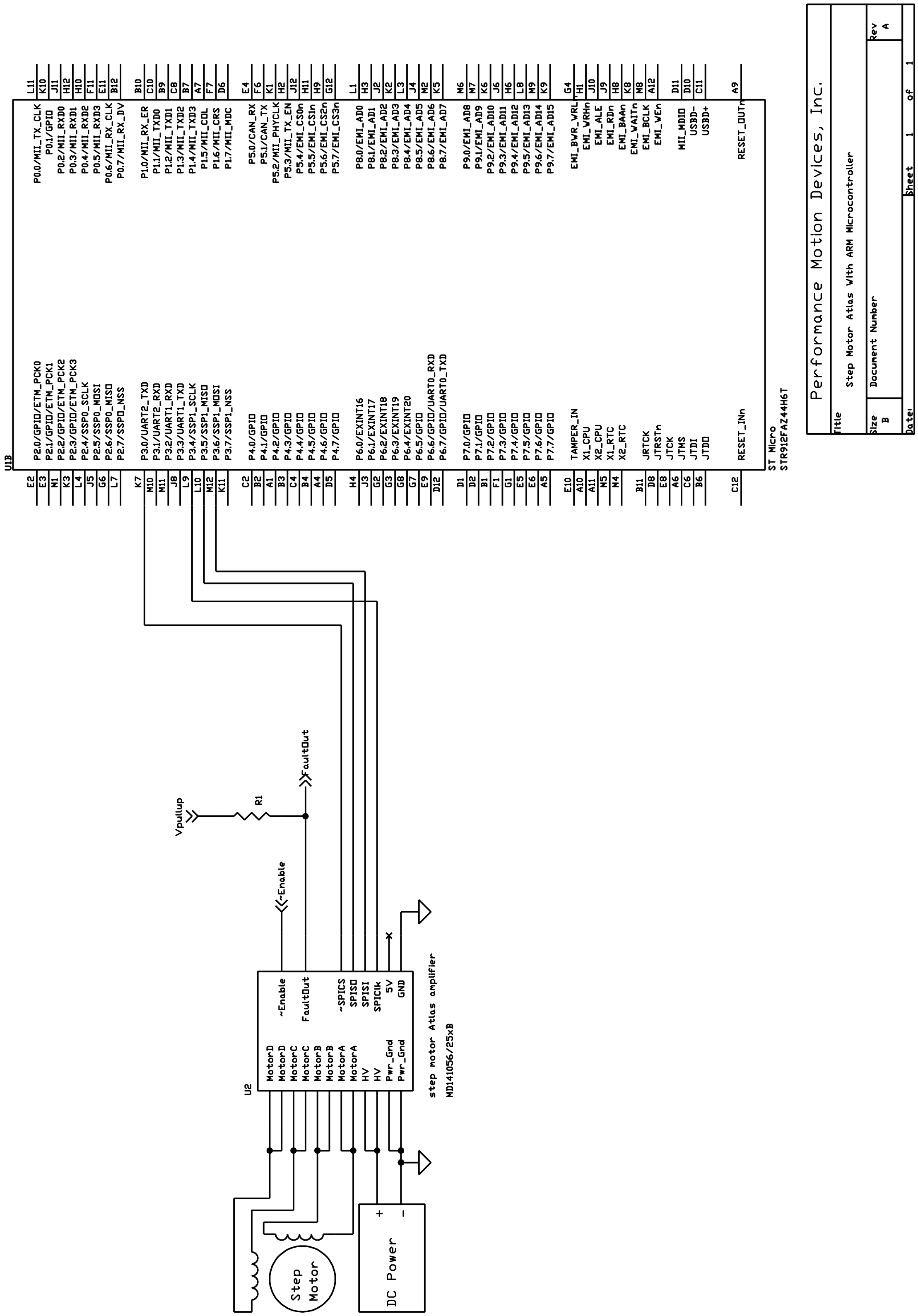
FaultOut is a dedicated output. During normal operation it outputs low. When a fault occurs it will go into a high impedance state. In this example, FaultOut is pulled up by Vpullup through resistor R1. Vpullup can be up to 24V to meet the system requirement. For example, if the fault signal is wired to a 5V TTL input, Vpullup can be 5V.

## B.6.4 Microcontroller Configuration

In this schematic, the host controller is ST's STR912FAZ44H6T. The microcontroller's SPI port is used for SPI communication.

In this example output pin K7 of the U1 (ARM processor) is used to control Atlas' ~SPICS input. ~SPICS has an internal pull-up, therefore, during power up and after reset, the control pin should be in high-impedance or output high state to disable the SPI.

**Figure B-5: Step Motor Atlas With ARM Microcontroller**



# B.7 Atlas Interfacing Via a Daughter Card

The following schematic shows an example of Atlas interfacing via a daughter card.

## B.7.1 Atlas Application Considerations Via a Daughter Card

Ground placement is critical for Atlas operation. Atlas is powered through pin pairs HV and Pwr_Gnd with a transformer-isolated DC power supply. The Pwr_Gnd and GND pins are shorted inside the Atlas, and at a system level they refer to the same ground. Pwr_Gnd is the current return path for the power train, while GND is the reference for the SPI signals and other digital control signals. Also, there is another current return path from the high frequency component of the motor winding current. To ensure optimal performance, star grounding is recommended for component placement and layout. That is, Pwr_Gnd and GND should be connected to the system ground very close to Atlas, and the ground return paths should be kept away from each other. Please refer to B.1 for general layout and wiring considerations on power input and motor outputs.

When Atlas is used with a daughter card, above grounding requirements might be difficult to implement at system level. For example, Atlas is installed close to the motor on a daughter card while the host controller resides in the mother board. The host controller controls Atlas through a cable, and too long a cable might compromise the module performance. Another example is that a single power supply powers multiple Atlas daughter card at different locations through long, separate, power cables. The long cables establish a current loop, and the ground current might interfere with normal Atlas operation.

This application note provides some examples to address above issues. In the example schematic, PMD's Magellan IO and CP chips are used to control the two Atlas units on the daughter card(s). Because of the length of the connecting cable between the host board and daughter board(s), there are buffers added on the SPI bus on the host board in order to boost the signal driving and sinking capabilities.

## B.7.2 Atlas SPI Through Isolator

Atlas receives control commands through an SPI interface and functions as an SPI slave; The SPI signals refer to its "local" ground. However, when Atlas is on a daughter card with a cable to the host controller, the host controller's local ground might be different, and SPI communication might see errors due to the ground difference/noise.

An isolator can be used to "break" the ground so that Atlas and the host controller refer to their own grounds. In this example, U1 and U2 are digital isolators. Atlas refers to "side B" ground and the host controller to "side A" ground. Please note that the isolators have to meet the timing specification for Atlas communication because the isolator will add delay to the signals. The buffer U5 and U6 are to boost the signal driving and sinking capability after the isolator output. In this example, DB9 connector with shielded DB9 cable is used. When R3 is zero, the shield is connected to side A ground at the daughter card end. Similarly, on the Magellan side (on the mother board), buffer U7 and U9 are used to boost the signal driving and sinking capability, and, when R4 is zero, the shield is connected to side A ground at the host end. Depending on the design layout, a Schmitt trigger input or standard termination practice might be necessary for the application with long cables.

## B.7.3 HV and Pwr_Gnd High-Frequency Isolation

When a single power supply powers multiple Atlas modules through long, separate, power cables, the long cables establish a current loop because they are shorted at both the power supply and the Atlas end. It will result in ground currents that might interfere with normal Atlas operation.

In this case, an L-C-L network can be used to provide high-frequency isolations among the modules. For example, for Atlas U3, C1 is between the Atlas HV and Pwr_Gnd. It serves as the bank capacitor for Atlas operation when necessary. L1 is between Atlas HV and power supply positive output. L2 is between Atlas Pwr_Gnd and power supply ground return. The current rating of L1 and L2 has to fit the Atlas operation current. L1 and L2 will bring in high impedance at high frequency to attenuate the ground current. A separated L-C-L network is used for Atlas U4 for optimum performance.

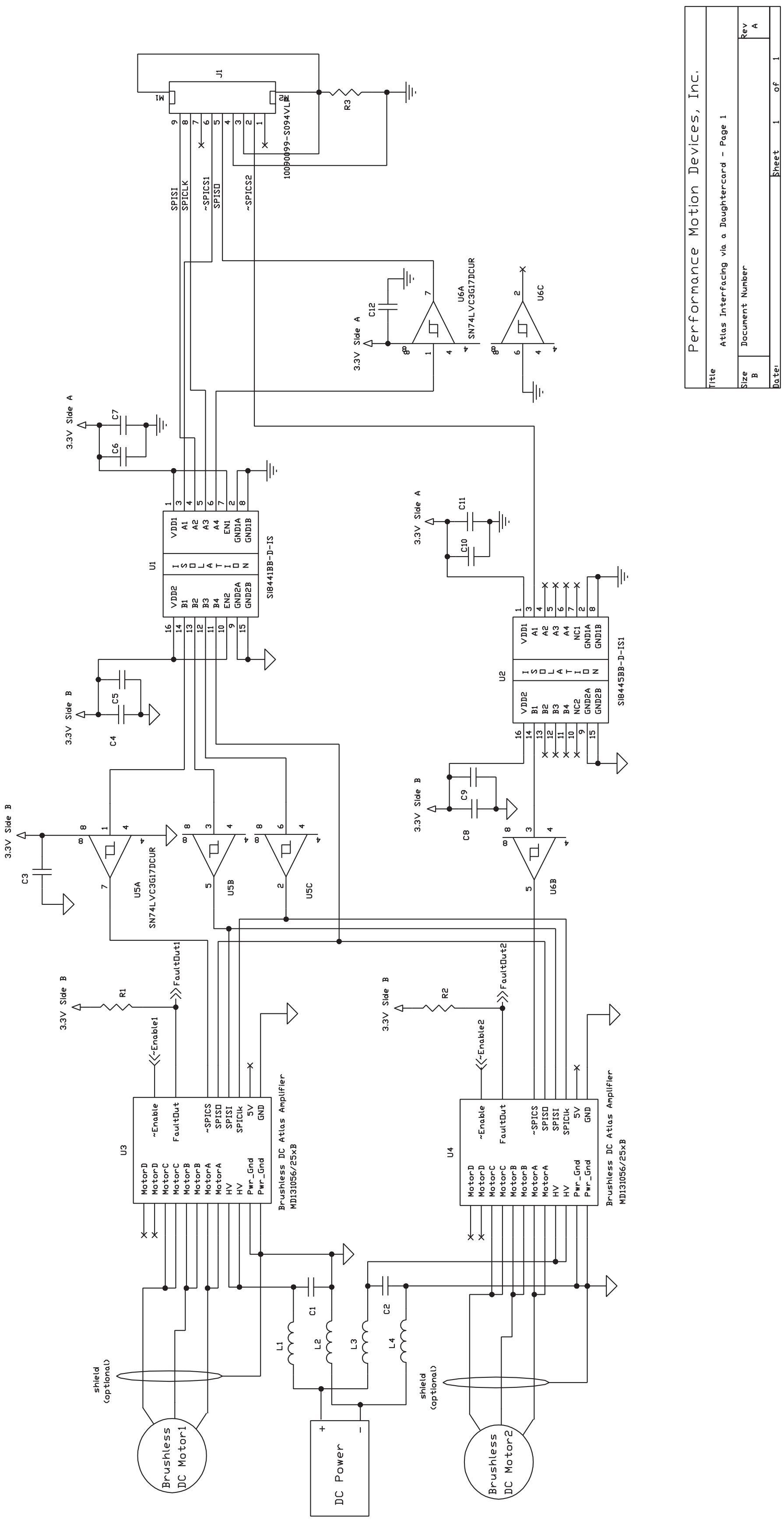
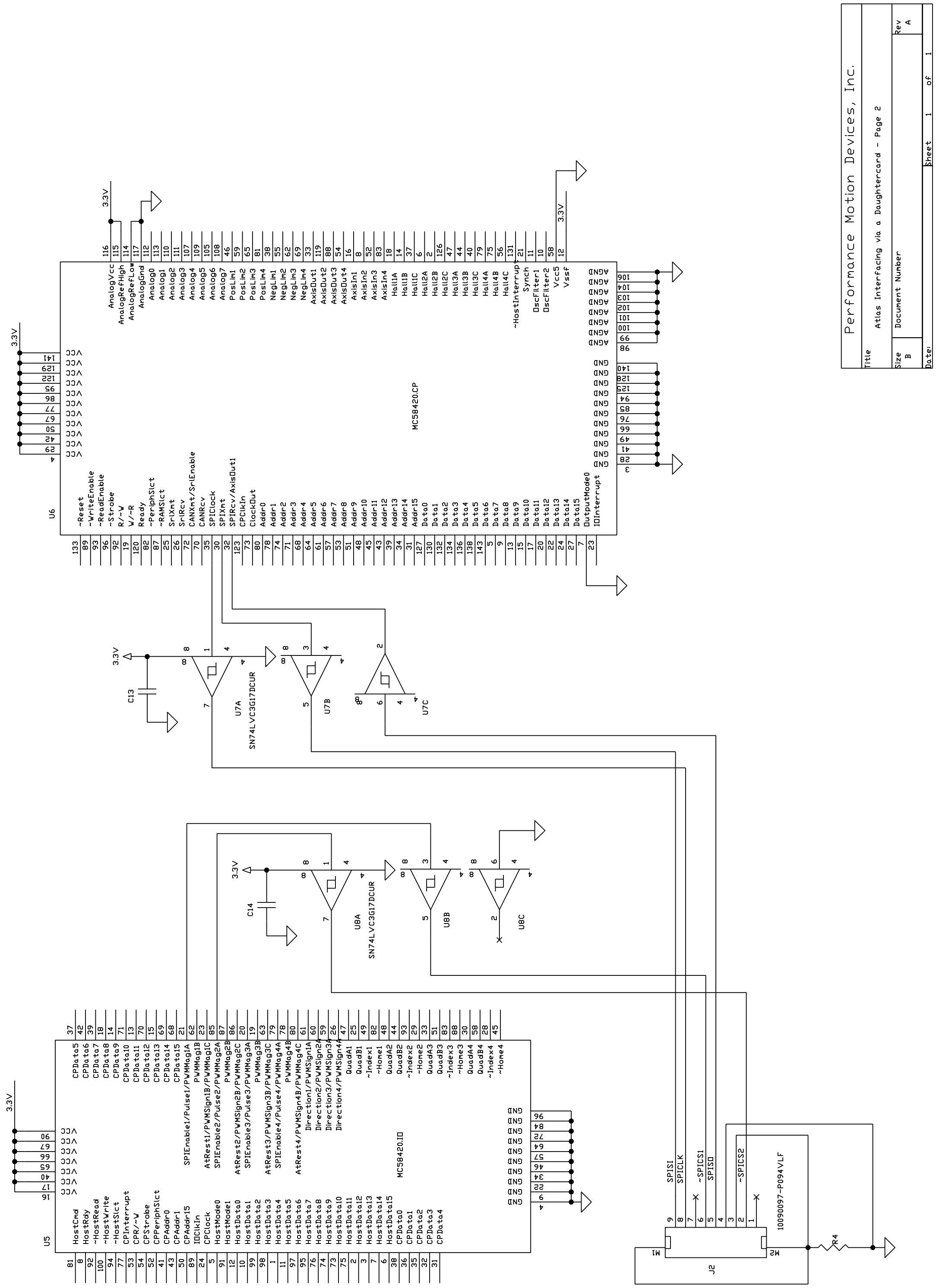



**Figure B-6: Atlas Interfacing Via A Daughter Card #1**

**Figure B-7: Atlas Interfacing Via A Daughter Card #2**



# B.8 Multi-Motor Atlas with Single-Axis MC58113 Motion Control IC

The following schematic shows multi-motor Atlas with single-axis MC58113 motion control IC.

## B.8.1 Atlas Power Input and Motor Outputs

Atlas is powered through pin pairs HV and Pwr_Gnd, and the power source is a transformer-isolated DC power supply.

Multi-motor Atlas can be configured to support DC Brush motor, Brushless DC motor or step motor. For DC Brush motors pins MotorA and MotorB are wired to motor windings Motor+ and Motor-, respectively. Pins MotorC and MotorD are left un-connected. For Brushless DC motors pin Motor A, MotorB and MotorC are wired to motor windings A, B and C, respectively. Pin MotorD is left un-connected. For step motors pins MotorA, MotorB, MotorC and MotorD are wired to motor windings A+, A-, B+ and B-,respectively.

Please refer to Section B.2, "Brushless DC Atlas With Single-Axis MC58113 Motion Control IC" for layout and wiring recommendations on power input and motor outputs.

## B.8.2 Atlas SPI Interface

Atlas receives control commands through an SPI interface and functions as an SPI slave. Atlas SPI communication is enabled when ~SPICS is pulled down. Only one Atlas can be enabled at any given time. Please refer to Section B.2, "Brushless DC Atlas With Single-Axis MC58113 Motion Control IC" for layout recommendation on SPI interface.

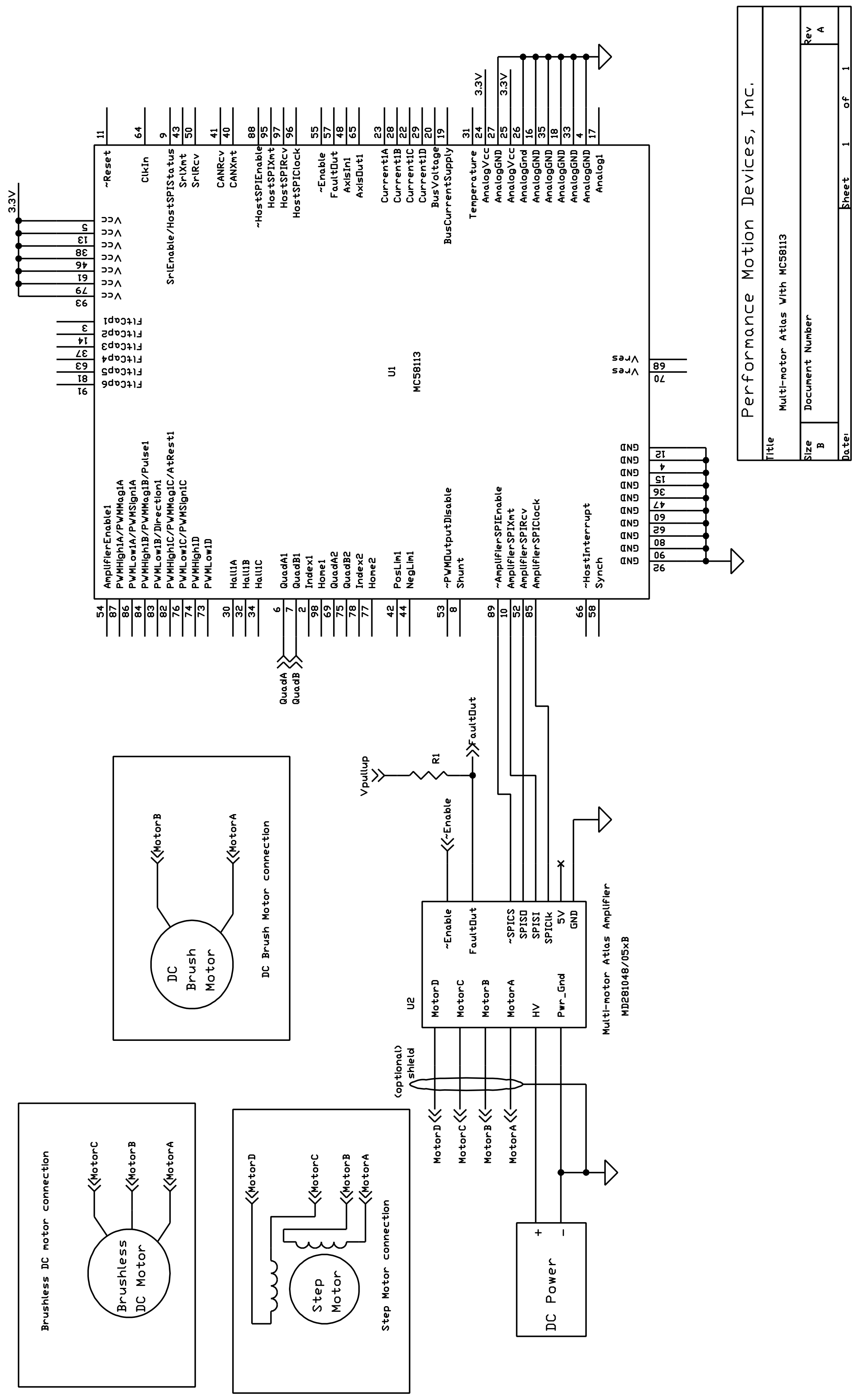
## B.8.3 Atlas ~Enable and FaultOut Signals

Atlas has one dedicated input signal, ~Enable, which must be pulled low for the Atlas output stage to be active. FaultOut is a dedicated output. During normal operation it outputs low. When a fault occurs it will go into a high impedance state. In this example, FaultOut is pulled up by Vpullup through resistor R1. Vpullup can be up to 24V to meet the system requirement. For example, if the fault signal is wired to a 5V TTL input, Vpullup can be 5V.

## B.8.4 Magellan MC58113 Configuration

In this schematic the SPI master is a single-axis MC58113. Only the connections with Atlas are shown. For complete Magellan wiring, please refer to the MC58113 electrical specifications.

**Figure B-8: Multi-motor Atlas With MC58113 Motion Control IC**



# Index

*This page intentionally left blank.*