PCI-8392/8392H

DSP based SSCNET III
Motion Control Card
User's Manual

Manual Rev.	2.00
Revision Date:	September 20, 2008
Part No:	50-11156-1000

Recycled Paper

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Advance Technologies; Automate the World.

Copyright 2008 ADLINK TECHNOLOGY INC.

All Rights Reserved.

The information in this document is subject to change without prior notice in order to improve reliability, design, and function and does not represent a commitment on the part of the manufacturer.

In no event will the manufacturer be liable for direct, indirect, special, incidental, or consequential damages arising out of the use or inability to use the product or documentation, even if advised of the possibility of such damages.

This document contains proprietary information protected by copyright. All rights are reserved. No part of this manual may be reproduced by any mechanical, electronic, or other means in any form without prior written permission of the manufacturer.

Trademarks

NuDAQ, NuIPC, DAQBench are registered trademarks of ADLINK TECHNOLOGY INC.

Product names mentioned herein are used for identification purposes only and may be trademarks and/or registered trademarks of their respective companies.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Getting Service from ADLINK

Customer Satisfaction is top priority for ADLINK Technology Inc. Please contact us should you require any service or assistance.

ADLINK TECHNOLOGY INC.

Web Site:	http://www.adlinktech.com
Sales & Service:	Service@adlinktech.com
TEL:	+886-2-82265877
FAX:	+886-2-82265717
Address:	9F, No. 166, Jian Yi Road, Chungho City, Taipei, 235 Taiwan

Please email or FAX this completed service form for prompt and satisfactory service.

Company Information	
Company/Organization	
Contact Person	
E-mail Address	
Address	
Country	
TEL	FAX:
Web Site	
Product Information	
Product Model	
Environment	OS: M/B: CPU: Chipset: BIOS:

Please give a detailed description of the problem(s):

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Table of Contents

List of Tables ..... **iv**

List of Figures ..... **v**

1 Introduction **1**
- 1.1 Hardware Features 5
  - Specifications 5
  - Environmental Conditions 8
- 1.2 Software Support 9
  - Programming Library 9
  - MotionCreatorPro 2 9
- 1.3 Available Terminal Board 9

2 Installation **11**
- 2.1 Package Contents 12
- 2.2 PCI-8392/8392H Outline Drawing 13
- 2.3 PCI-8392/8392H Hardware Installation 15
  - Hardware Configuration 15
  - PCI Slot Selection 15
  - Installation Procedures 15
  - Troubleshooting 16
- 2.4 Software Driver Installation 17
- 2.5 Pin Assignments 17
  - RJ1: HSL Port Connector (PCI-8392H Only) 17
  - EMG1: EMG Signal Connector 17
- 2.6 SW1 Switch Setting for Card Index 18

3 Signal Connections **19**
- 3.1 SSCNet III Servo Driver Connection 19
- 3.2 RJ1: HSL Pin Assignment (PCI-8392H Only) 20
- 3.3 EMG Signal Connection 21

4 Operation Theory **23**
- 4.1 Architecture 24
  - HOST PC and SSCNET Board 24
  - SSCNet Communication 25
- 4.2 Single Motion 26
  - Single Axis Velocity Motion 26

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Single Axis P-to-P Motion 28
Linear Interpolation 30
Circular Interpolation 38
Speed Override 40
Position Override 43
4.3 Home Move 46
4.4 Jogging 52
4.5 Point Table 58
Point Table Construction 58
Point index 61
Point table execution 62
4.6 Motion Status and Related IO Monitoring 73
Position Control and Feedback 74
Motion I/O Status 75
Motion Status 77
4.7 Driver Management 78
Driver parameter 78
Servo On 79
Servo Alarm 79
4.8 Data Sampling 80
4.9 Torque Limit 88
4.10 Interrupt Control 91
4.11 Absolute Position System 94
4.12 E-Gear 96
4.13 DSP-based Gantry Function 98
Closed-loop Gantry Mode 99
Error Handling of Gantry Mode 100
4.14 HSL System Introduction (PCI-8392 Only) 101
The HSL System 101
HSL System Specifications 104
HSL Series Products 106

5 MotionCreatorPro 2 107
5.1 Run MotionCreatorPro 2 107
5.2 About MotionCreatorPro 2 107
5.3 MotionCreatorPro 2 Introduction 108
Main Menu 108
Card Number Layer 112
Axis/Module Layer 113
Servo Parameter Configuration 114

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Axis/Board Configuration 115
Single Movement 116
Home Return 118
Interpolation 120
Sampling 121
5.4 HSL Module Function Test 122
Module Type1: HSL-DI16DO16-N/P 122
Module Type2: HSL-DI32-N/P 122
Module Type2: HSL-DO32-N/P 122

6 Function Library 123
6.1 C/C++ Programming Library 123
6.2 List of Functions 124
6.3 Initialzation 127
6.4 SSCNET function 139
6.5 Motion IO and motion status 156
6.6 Single axis motion 164
6.7 Interpolation 180
6.8 Interrupt 190
6.9 Sampling 205
6.10 Point table 211
6.11 Field bus 229
6.12 Definition Tables 240
Board Parameters Definition Table 241
Axis Parameter Definition Table 242
SSCNET Axis Parameters Definition Table 244
Sampling Parameters Definition Table 245
Sampling Source Definition Table 246
Motion IO Status Bit Definition 247
Motion Status Definition Table 248
Interrupt Factor Table 250
Function Return Code 252
6.13 Error Code Table 253

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

List of Tables

Table 1-1: Specifications 5
Table 1-2: Environmental Conditions 8
Table 2-1: RJ1: HSL Port Connector 17
Table 2-2: EMG1: EMG Signal Connector 17
Table 2-3: SW1 Switch Setting for Card Index 18
Table 3-1: Fiber Lengths 19
Table 3-2: RJ1: HSL Pin Assignment (PCI-8392H Only) 20
Table 4-1: Axis Parameter Settings for ORG Home Moves 46
Table 4-2: Axis Parameter Settings for EZ Home Moves 50
Table 4-3: POINT_DATA Structure 59
Table 4-4: Motion IO Status Bit Definition 76
Table 4-5: Motion status definition table 77
Table 4-6: Sampling Parameters Definition Table 81
Table 4-7: Sampling Source Definition Table 82
Table 4-8: SSCNET Axis Parameters Definition Table 89
Table 4-9: Interrupt Factor Table 92
Table 4-10: Remote Operation 104
Table 4-11: Slave I/O modules 106
Table 4-12: Terminal Base 106
Table 6-1: Board Parameters Definition Table 241
Table 6-2: Axis Parameter Definition Table 243
Table 6-3: SSCNET Axis Parameters Definition Table 244
Table 6-4: Sampling Parameters Definition Table 245
Table 6-5: Sampling Source Definition Table 246
Table 6-6: Motion IO Status Bit Definition 247
Table 6-7: Motion Status Definition Table 249
Table 6-8: Interrupt Factor Table 251

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

List of Figures

Figure 1-1: Block Diagram of the PCI-8392/8392H 3
Figure 1-2: Flow Chart for Building an Application..... 4
Figure 2-1: PCI-8392 PCB Layout 13
Figure 2-2: PCI-8392H PCB Layout..... 14
Figure 3-1: Wiring for SSCNET III Configuration 19
Figure 3-2: Fiber Cable Appearance..... 19
Figure 4-1: HSL Overview..... 102

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

1 Introduction

The PCI-8392 is a PCI bus interface card designed for personal computer sor industrial computer accompanied with a Mitsubishi MR-J3B type or SSCNET III servo amplifier. The PCI-8392 can control up to 16 servo amplifiers.

The PCI-8392H is a PCI bus interface card that controls up to 16 SSCNET axes and single HSL network ports in one board.

The connection between the motion control board and the amplifier is performed via a high-speed serial communication using the SSCNet III protocol. SSCNet III connections offer the following advantages over pulse train connections:

- Wiring is simplified as servo amplifiers are connected via a multi-drop method and the communication distance is up to 50 meters.
- Parameter management and the construction of absolute positioning system (ABS) are greatly simplified.
- Since commands are transmitted in serial data format, noise-reduction is improved and thus reliability is improved. In addition, the control resolution is increased.
- An abundant amount of information can be retrieved from the servo system through SSCNet III. No longer are you restricted to commands and feedback. You can now also monitor servo status, alarm status and tuning servo parameters.

The PCI-8392/8392H uses the latest DSP and PC-based technology to improve the machine productivity and it also provides many kinds of user friendly developing interfaces to reduce the developing time.

The motion kernel inside PCI-8392/8392H has the most basic motion element that can be implemted for many kinds of application the ASIC-based motion controller cannot do. The PCI-8392/8392H uses closed-loop control architechture to ensure the precision in postioning purpose. Unlike the pulse-train motion controller, the PCI-8392/8392H will feedback the real controlling position and loop back to DSP for compensating next position. The pulse-train motion controller can only take care about the precision on

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

motor itself. If the moving parts which on the stage has position error occured, it does not know at that moment. The pulse-train type controller is something like blind control or open-loop control in positioning.

The gain tuning in a closed-loop system is very important on performance. The PCI-8392/8392H provides a utility, MotionCreator-Pro, to reduce the efforts on gain tuning. For uncertain loading in motor system, we will have advanced auto tuning gain system in advanced DSP kernel.

Integrated with HSL technology support, PCI-8392H offers one HSL port to increase distributed I/O requirement on PCI-8392H in Machine System. HSL is an innovative distributed I/O technology, which allows thousands of I/O points to be scanned in millisecond-level real time by using mater-slave architecture. By using commercial Ethernet cable with RJ45 connector, users may easily set up the HSL slaves modules as close as possible to sensor devices. Thus the wiring effort is dramatically reduced. A variety of machine makers can benefit from HSL network because it integrates discrete I/O, analog I/O, and thermocouple module. This local network features rapid response, real-time scanning.

Figure 1-1 shows the functional block diagram of the PCI-8392/8392H card. It contents a DSP as a main chip. Around the DSP is the main SSCNET III ASIC and some necessary devices for the DSP such as SDRAM and DPRAM.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

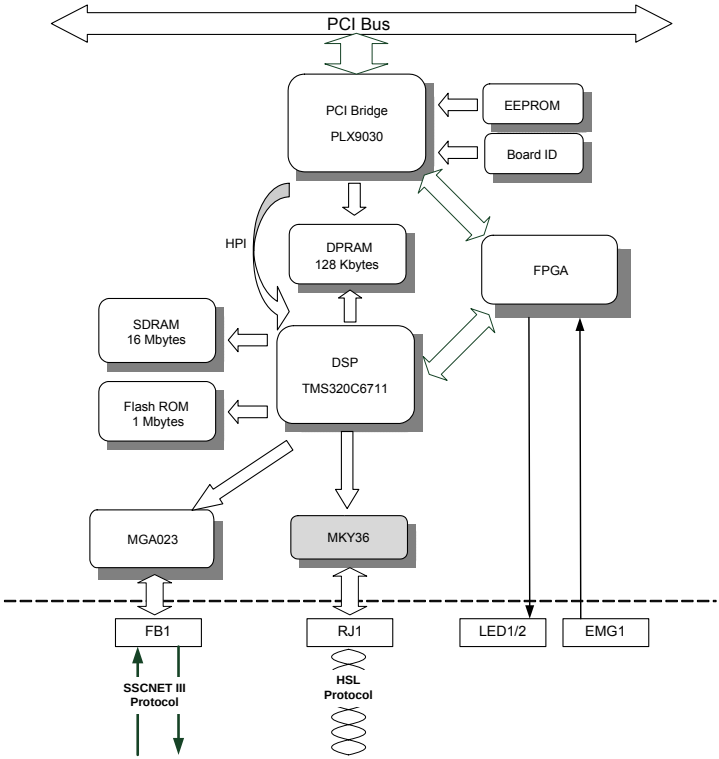


Figure 1-1: Block Diagram of the PCI-8392/8392H

MotionCreatorPro 2 is a Windows-based application development software package included with the PCI-8392/8392H. MotionCreatorPro 2 is useful for debugging a motion control system during the design phase of a project. An on-screen display lists all installed axes information and I/O signal status of the PCI-8392/8392H.

Windows programming libraries are also provided for C++ compiler, Visual Basic and many other popular languages. Sample programs are provided to illustrate the operations of the functions.

Figure 1-2 illustrates a flow chart of the recommended process in using this manual in developing an application. Refer to the related chapters for details of each step.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

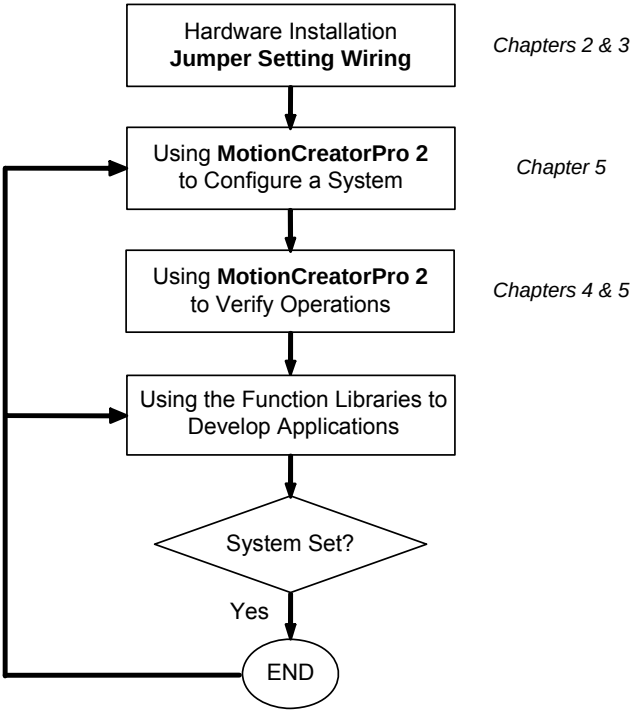


Figure 1-2: Flow Chart for Building an Application

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

1.1 Hardware Features

- 16-axis DSP-based SSCNET III motion control board with PCI interface
- One built-in HSL bus which controls HSL I/O and stepper modules
- 0.888 ms servo update rate for 16 axes and 0.444 ms servo update rate for 8 axes.
- 4-bit Board ID for multiple boards indexing
- On-board flash ROM for your non-volatile data
- One independent emergency input pin
- Board to board synchronous bus extension bus for additional customized functions

1.1.1 Specifications

	Item	Description
System	Bus Type for PCI board	PCI Rev. 2.2, 33 MHz
	Bus width for PCI	32-bit
	Bus Voltage	3.3 V, 5 V
	Memory usage	2 M x 32 bit
	IRQ on PCI board	Assigned by PCI controller
General Specifications	Operating temperature	0°C to 60°C
	Storage temperature	-20°C to 80°C
	Humidity	5 - 95%, non-condensing
	Power Consumption	PCI-8392/8392H: +5 V @ 1 A typical
DSP	Type	TI TMS320C6711D
	Clock	250 MHz
	DSP performance	1500 MFLOPS
Board Interface	SSCNet Connector	One fiber connector
	HSL Connector	One RJ45 connector

Table 1-1: Specifications

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

	Item	Description
Driver Communication	Protocol	SSCNET III
	Reference Clock	50 Mhz
	Maximum working length	50 m for each 16 axes
	Error detection	CRC
HSL Communication	Maximum working length	2400 m (with repeater use)
	Transfer baud rate	3/6/12 MHz
Servo Loop	Max. No of controllable axes	16
	Servo update rate	0.888 ms for 16 Axes
		0.444 ms for 8 Axes
	Servo Data Monitors	Current position
		Droop (deviation)
		Velocity Command
		Velocity feedback
		Torque command
		Servo alarm number …etc
	Servo parameter tuning	Parameter read/write
Motion Functions	Motion Velocity Profile	Trapezoidal & S-Curve
	Single motion	Jog move
		Single axis P to P motion
		Change P/V on the fly
		Linear interpolation: up to 4 axes
		2-axis Circular interpolation
	Home move	1 home mode
	Point table motion	Start / End motion list
		Add linear trajectory
		Add arc trajectory: 2 axes
		Add Dwell
		Start/Sop command
	Motion status monitor	Motion IO status read/configure
		Motion status

Table 1-1: Specifications

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

	Item	Description
Application Functions	Gantry function	
	E-Gear function	
	Data sampling	
	System error check	Watchdog timer
Interrupt	During operation stop	Possible to select conditions where interrupt occurs
	During alarms, etc.	Yes

Table 1-1: Specifications

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

1.1.2 Environmental Conditions

Item	Specification				
Ambient Temperature Operation	0-55°C				
Ambient Temperature Storage	-20-75°C				
Ambient Humidity Operation	10-90%RH, avoid condensation				
Ambient Humidity Storage	10-90%RH, avoid condensation				
Vibration Resistance	Confirms to JIS C 0911	Frequency	Acceleration	Amplitude of Vibration	Sweep
		10-55 Hz	-	0.075 mm	10 times*
		55-150 Hz	1G	-	(1 ctave/minute)
Shock Resistance	Confirms to JIS C 0912 (10 g, 3 directions, 3 times)				
Noise Resistance	Noise voltage 1500 V.P.P, Noise frequency 25-60 Hz using noise simulator				
Operating Atmosphere	Minimal corrosive gas, dust				
Cooling Method	Self-cooling				

Table 1-2: Environmental Conditions

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

1.2 Software Support

1.2.1 Programming Library

For customers who are programming their own applications, we provide Windows 2000/XP/Vista DLL for the PCI-8392/8392H. It is shipped with these boards.

1.2.2 MotionCreatorPro 2

MotionCreatorPro 2 is a Windows-based utility to setup cards, motors and system. It can also help users debug hardware and software problems. It also can let users to setting the I/O logic parameters, which can be loaded in their own program. This product is bundled with this card.

Refer to Chapter 5 for details.

1.3 Available Terminal Board

ADLINK provides the DIN-839-J3B terminal board for servo driver I/O, such as alarm signal, in-position signal, analog monitor singal and general purposed digital input signal.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2 Installation

This chapter describes how to install the PCI-8392/8392H. Please follow these steps below:

- Check what you have (Section 2.1, page 12)
- Check the PCB (Section 2.2, page 13)
- Install the hardware (Section 2.3, page 15)
- Install the software driver (Section 2.4, page 17)
- Understanding the I/O signal connections (Chapter 3, page 19) and their operations (Chapter 4, page 23)
- Understanding the connector pin assignments (the remaining sections) and wiring the connections

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.1 Package Contents

In addition to this User's Guide, the package also includes the following items:

- PCI-8392/8392H: SSCNET III 16-axis Motion Controller
- ADLINK All-in-one Compact Disc
- Terminal board is optional accessory. This would not be included in PCI-8392/8392H package.

If any of these items are missing or damaged, contact the dealer from whom you purchased the product. Save the shipping materials and carton to ship or store the product in the future.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.2 PCI-8392/8392H Outline Drawing

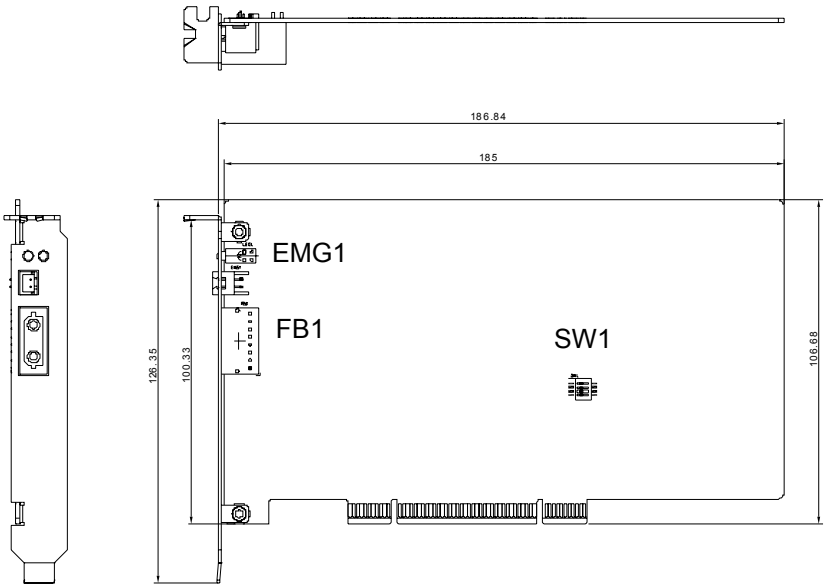


Figure 2-1: PCI-8392 PCB Layout

FB1: Main signal connector (fiber interface)

EMG1: Emergency input signal connector (2-pin)

SW1: DIP switch for card index selection (0-15)

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

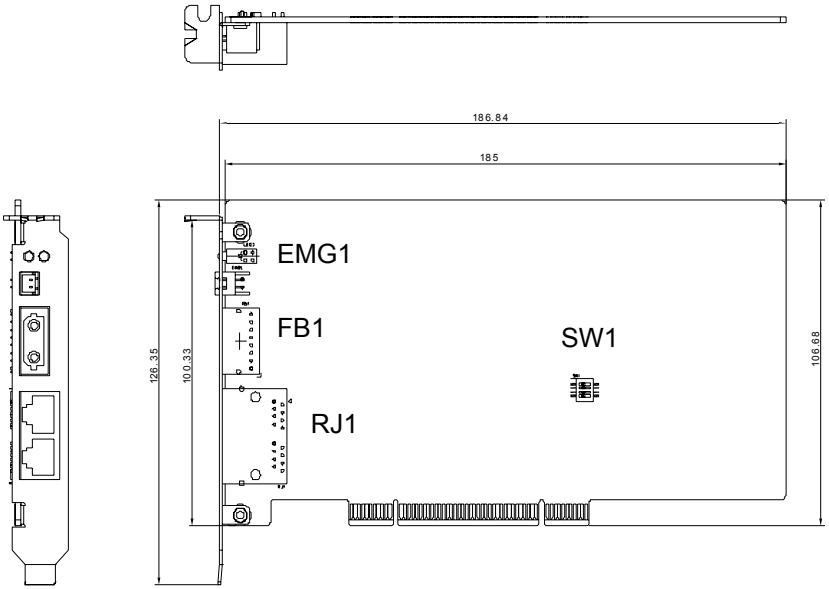


Figure 2-2: PCI-8392H PCB Layout

FB1: Main signal connector (Fiber interface)

EMG1: Emergency input signal connector (2-pin)

RJ1: HSL main connector (RJ45)

SW1: DIP switch for card index selection (0-15)

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.3 PCI-8392/8392H Hardware Installation

2.3.1 Hardware Configuration

The PCI-8392/8392H is fully Plug and Play compliant. Hence memory allocation (I/O port locations) and IRQ channel of the PCI card are assigned by the system BIOS. The address assignment is done on a board-by-board basis for all PCI cards in the system.

2.3.2 PCI Slot Selection

Your computer system may have both PCI and ISA slots. Do not force the PCI card into a PC/AT slot. The PCI-8392/8392H can be used in any PCI slot.

2.3.3 Installation Procedures

1. Read through this manual and setup the jumper according to your application
2. Turn off your computer. Turn off all accessories (printer, modem, monitor, etc.) connected to computer. Remove the cover from your computer.
3. Select a 32-bit PCI expansion slot. PCI slots are shorter than ISA or EISA slots and are usually white or ivory.
4. Before handling the PCI-8392/8392H, discharge any static buildup on your body by touching the metal case of the computer. Hold the edge of the card and do not touch the components.
5. Position the board into the PCI slot you have selected.
6. Secure the card in place at the rear panel of the system unit using screws removed from the slot.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.3.4 Troubleshooting

If your system does not boot or if you experience erratic operation with your PCI board in place, it is most likely caused by an interrupt conflict (possibly an incorrect ISA setup). In general, the solution, once determined it is not a simple oversight, is to consult the BIOS documentation that comes with your system.

Check the control panel of the Windows system if the card is listed by the system. If not, check the PCI settings in the BIOS or use another PCI slot.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.4 Software Driver Installation

1. Auto run the ADLINK All-In-One CD. Choose Driver Installation -> Motion Control -> PCI-8392/8392H
2. Follow the procedures of the installer.
3. After setup installation is completed, restart windows.

Suggestion: Please download the latest software from ADLINK website if necessary.

2.5 Pin Assignments

2.5.1 RJ1: HSL Port Connector (PCI-8392H Only)

Pin No	Signal Name
1	NC
2	NC
3	RX+
4	TX-
5	TX+
6	RX-
7	NC
8	NC

Table 2-1: RJ1: HSL Port Connector

2.5.2 EMG1: EMG Signal Connector

Pin No	Signal Name
1	EGND
2	EMG

Table 2-2: EMG1: EMG Signal Connector

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2.6 SW1 Switch Setting for Card Index

The SW1 switch is used to set the card index. For example, if you turn 0 to ON and others are OFF. It means the card index as 0. The value is from 0 to 15. Refer to the following table for details.

Card ID	Switch Setting (ON=0)
0	1111
1	1110
2	1101
3	1100
4	1011
5	1010
6	1001
7	1000
8	0111
9	0110
10	0101
11	0100
12	0011
13	0010
14	0001
15	0000

Table 2-3: SW1 Switch Setting for Card Index

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3 Signal Connections

Signal connections of all I/O's are described in this chapter. Refer to the contents of this chapter before wiring any cables between the 8392 (PCI-8392H) and any motor drivers

This chapter contains the following sections:

- Section 3.1: SSCNet III Servo Driver Connection
- Section 3.2: RJ1: HSL Pin Assignment (PCI-8392H Only)
- Section 3.3: EMG Signal Connection

3.1 SSCNet III Servo Driver Connection

Wiring for 16 Axes

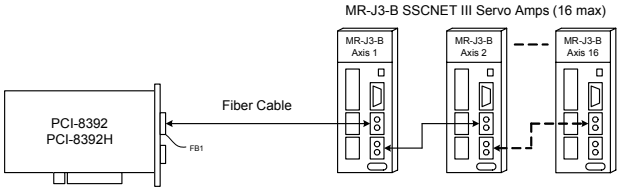


Figure 3-1: Wiring for SSCNET III Configuration

SSCNet III Cable

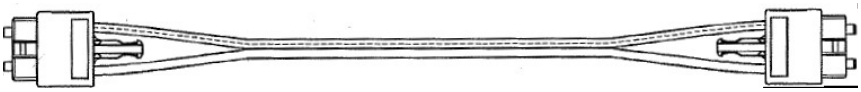


Figure 3-2: Fiber Cable Appearance

Fiber Lengths Offered by ADLINK

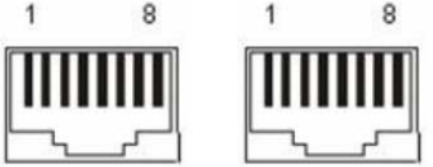
Fiber Length (Unit: m)
0.15
0.30
0.50
1.00
3.00
5.00

Table 3-1: Fiber Lengths

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.2 RJ1: HSL Pin Assignment (PCI-8392H Only)

The HSL master is the key component in charge of communicating with slave I/O modules. The master sets output values to and gathers input information from slaves. PCI-8392H provides two ports of HSL master connection for more flexibility wiring. Pin assignment on RJ1 of PCI-8392H lists as below:



Pin No.	Pinout
1	NC
2	NC
3	RX+
4	TX-
5	TX+
6	RX-
7	NC
8	NC

Table 3-2: RJ1: HSL Pin Assignment (PCI-8392H Only)

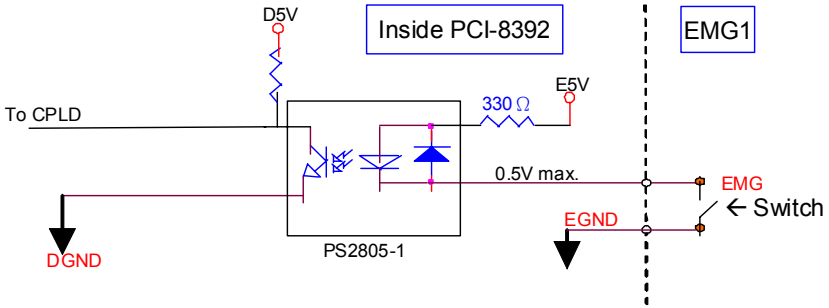
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3.3 EMG Signal Connection

The EMG signal is used as input signal for the emergency stop of the motor. The following table lists signal names, pin numbers:

Pin No	Signal Name
1	EGND
2	EMG

The input circuit of the EMG signal is shown below. Usually, a switch is used to indicate the EMG occurs. The specifications of the limit switch should have contact capacity of +5 V @ 10mA minimum.



EMG logic can be changed through software settings. Please refer to Section 6.12.1, page 241.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4 Operation Theory

This chapter describes the detail operation of the SSCNET board card. Contents of the following sections are as follows.

- Section 4.1: Architecture
- Section 4.2: Single Motion
- Section 4.3: Home Move
- Section 4.4: Jogging
- Section 4.5: Point Table
- Section 4.6: Motion Status and Related IO Monitoring
- Section 4.7: Driver Management
- Section 4.8: Data Sampling
- Section 4.9: Torque Limit
- Section 4.10: Interrupt Control
- Section 4.11: Absolute Position System
- Section 4.12: E-Gear
- Section 4.13: DSP-based Gantry Function

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.1 Architecture

4.1.1 HOST PC and SSCNET Board

The communication between the host PC and the SSCNET board is through a 128 Kbyte dual port RAM that is integrated inside the SSCNET board. Both the Host CPU and DSP can read/write on it.

On a hardware level, the SSCNET board is a small microcomputer. It has its own processor (the DSP), address and data bus, its data memory, and peripherals use for SSCNet communication protocol. Thus, the host CPU does not require the constant attention of the DSP. The only time the host needs to perform the read/write functions to the DPRAM is when the application program requests information or sends command motions.

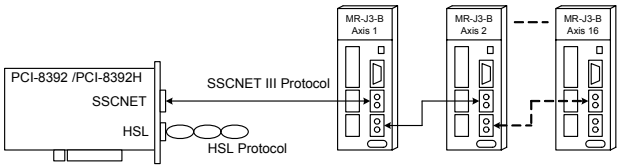
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.1.2 SSCNet Communication

The SSCNET board controls servomotors through the SSCNet communication. The communication is master-slave architecture. Every 0.444ms/0.888 ms, the SSCNET Board (master) sends a command, in which the position command is involved, to each servo driver (the slave) and in return the servo drivers report back to the SSCNET board, providing the SSCNET board with information about its position, velocity, and other specified servo data.

Control Cycle	Maximum Number of Axes Controlled
0.444 ms	8 axes
0.888 ms	16 axes

- Communication is synchronizes with the control cycle of the controller and the transmit/receive is executed every control cycle
- Broadcast transmission is conducted from the SSCNET board to the servo amplifier
- Transmission from the servo driver to the SSCNET board is conducted through a time-division system, and SSCNET board reads data in a batch format.



LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.2 Single Motion

In this section, single motion functions are discussed. Single motion means the motion is commanded by one function call only. For example, **APS_relative_move**(), this function will allow an axis to move a certain distance with a specified speed and other motion parameter, such as accel/decel time and so on.

In this manual, in order to reduce and manage the programming of motion operation, therefore, user has to config whole motion type and parameter via function **APS_set_axis_param**() in arrear of motion operation firstly.

Single motion functions can be categorized into the following types according to their functionality.

- Section 4.2.1: Single Axis Velocity Motion
- Section 4.2.2: Single Axis P-to-P Motion
- Section 4.2.3: Linear Interpolation
- Section 4.2.4: Circular Interpolation
- Section 4.2.5: Speed Override
- Section 4.2.6: Position Override

4.2.1 Single Axis Velocity Motion

Veloctiy mode means the motion command is continuously outputting until a stop command is issued. The motor will run without a target position or desired distance unless it was stopped by other reasons. The output command profile will accelerate from a starting velocity to a specified maximum velocity. It can be follow a linear or S-curve acceleration shape. The command output rate is kept at maximum velocity until another velocity command overrided or a stop command issued. The velocity could be overrided by a new speed setting.

In this section, the following functions are discussed.

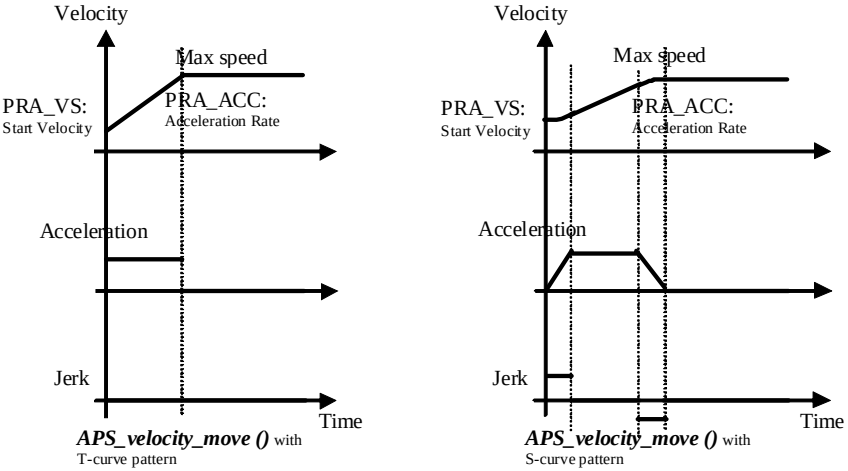
```
APS_set_axis_param(Axis_ID, AXS_Param_No,
     AXS_Param)
APS_velocity_move(Axis_ID, Max_speed)
```

The single axis velocity motion function will allow the axis to accelerate from a starting velocity, '**PRA_VS**', to a specified constant

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

velocity, 'Max_speed'. The axis will continue to travel at this constant velocity until the velocity is changed by overwiriting the function **APS_velocity_move**() or stopped by the functions **APS_stop_move**(), **APS_emg_stop**().

Two kinds of acceleration method are available. By using T-curve speed pattrn, the acceleration is constant as shown in the in left diagram below. By using S-curve speed pattrn, the derivative of acceleration, the '**jerk**', is a constant as Illustrated in the right diagram below. The speed profile of this kind of motion is shown as below:



Note: Please refer to Section 6.12: Definition Tables to set the axis parameter.

PRA_VS means start velocity, **PRA_VE** means end velocity, and **PRA_ACC/DEC** means acceleration or deceleration.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.2.2 Single Axis P-to-P Motion

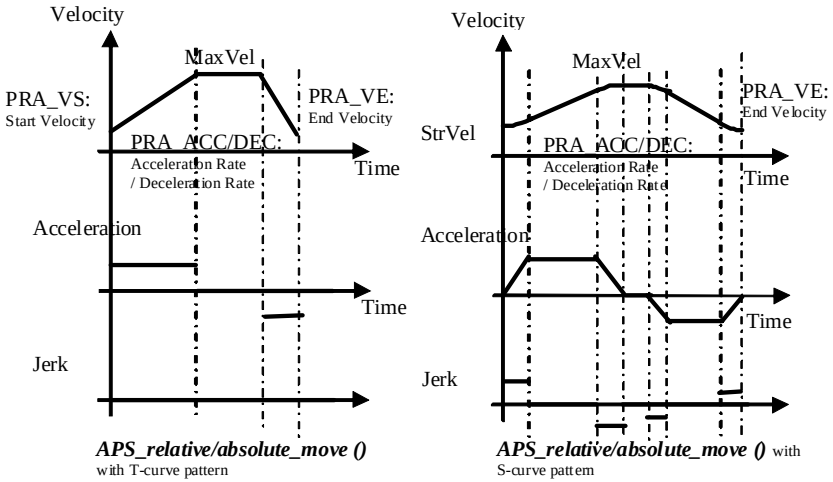
In this section, the following functions are discussed.

```
APS_relative_move(Axis_ID, Distance, Max_speed)
APS_absolute_move(Axis_ID, Position, Max_speed)
```

Single axis P-to-P motion functions will allow the axis to move a specified distance or move to a specified position. Above two functions are pretty straightforward. 'Relative' and 'absolute' characterizes the function and provides information about the position method to achieve the target position. States like upside section, user is able to define the velocity profile pattern, start velocity, acceleration / deceleration rate before, which via using function **APS_set_axis_param**().

If the velocity profile applied is '**Trapezoidal**'. That is the acceleration and deceleration is a constant (shown in left diagram).

If the velocity profile applied is '**S-Curve**'. This is a derivative of acceleration, 'jerk', and is a constant (shown in right diagram).



Note: Please refer to section 6.11 – "Definition table" to set the axis parameter.

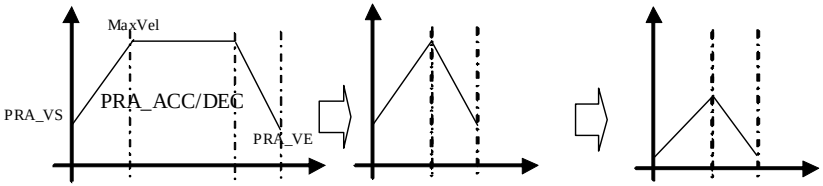
PRA_VS means start velocity, **PRA_VE** means end velocity, and **PRA_ACC/DEC** means acceleration or deceleration.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

The distance moved during acceleration and deceleration can be calculated using the following formula. (For both trapezoidal and S-curve profiles)

```
Dist_acc = 0.5 * (StrVel + MaxVel) * Tacc
Dist_dec = 0.5 * (FinVel + MaxVel) * Tdec
```

In some cases, the distance moved may not be long enough. For example, the '**Distance**' in **APS_relative_move**() is too small or '**Position**' in **APS_absolute_move**() is too close to the current position. These two function calls mentioned above automatically slows down the velocity. The change in the velocity profile is illustrated in the diagram below.



Case 1:
The distance is longer then acceleration and deceleration needed.

Case 2:
The distance is equal to acceleration and deceleration needed.

Case 3:
The distance is smaller than acceleration and deceleration needed.

Case 1 to Case 2: The constant velocity period is reduced while the **PRA_VS**, **PRA_VE**, **Max. velocity** and **PRA_ACC/DEC** remain unchanged.

Case 2 to Case 3: The constant velocity period vanished, and, **PRA_VS**, **PRA_VE**, **Max. velocity** become smaller according to the ratio described below. While **PRA_ACC/DEC** remains unchanged.

```
New_PRA_VS = K * Original_ PRA_VS
New_Max speed= K * Original_Max speed
New_PRA_VE = K * Original_PRA_VE
```

Where K = Dist/(Distacc_needed + Distdec_needed)= Dist/(Distjust_case)

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.2.3 Linear Interpolation

“Interpolation between multi-axes” means these axes start simultaneously, and reach their ending points at the same time. Linear means the ratio of speed of every axis is a constant value. In this section, the following functions are described.

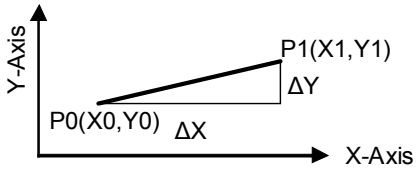
```
APS_absolute_linear_move (Dimension,
     Axis_ID_Array, Position Array,
     Max_Linear_speed)
APS_relative_linear_move (Dimension,
     Axis_ID_Array, Distance Array,
     Max_Linear_speed)
APS_absolute_arc_move (Dimension, Axis_ID_Array,
     Center_Pos_ Array, Max_Arc_speed, Angle)
APS_relative_ arc _move (Dimension,
     Axis_ID_Array, Center_Offset_ Array,
     Max_Arc_speed, Angle)
```

These four functions applied to any 2, any 3 or any 4 of the 16 axes in one card, so that these axes can “start simultaneously, and reach their ending points at the same time” and the ratio of speed between these axes is a constant value. Because the speed parameter is in vector direction within parameter table setting, this function will take the master axis’s acceleration and deceleration time constant to calculate. The master axis is the minor axis number that user perform an interpolation.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2 Axes Linear Interpolation

As in the diagram below, 2 axes linear interpolation means to move the XY (or any 2 of the 4 axis) position from P0 to P1. The 2 axes start and stop simultaneously, and the path is a straight line.



The speed ratio along X-axis and Y-axis is (ΔX: ΔY), respectively, and the vector speed is:

$$\frac{\Delta P}{\Delta t} = \sqrt{(\frac{\Delta X}{\Delta t})^2 + (\frac{\Delta Y}{\Delta t})^2}$$

When calling the two-axes linear interpolation functions, it is the **vector speed** to define the start velocity (master axis), **PRA_VS**, and maximum velocity, **Max_Linear_speed**, both trapezoidal and S-curve profile are available.

Example:

```
//…Initialize card
I32 Dimension = 2;
I32 Master_Axis_ID = 1; //Master axis
I32 Axis_ID_Array[4] = {1, 2}; //Axis ID 1 is
     master axis.
I32 Position_Array [4] = {1000, 2000}; //(Unit:
     pulse)
```

This cause the two axes (axes 1 & 2) to perform a linear interpolation movement, in which:

```
ΔX = 1000 pulse
ΔY = 2000 pulse
I32 Max_Linear_Speed = 1000;  //(Unit: pulse/
     second)
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

This way sets the maximum speed to 1000 pulse per second during performing 4-axes interpolation motion.

```
I32 Ret;
APS_set_axis_param(Master_Axis_ID, PRA_CURVE, 0);
     //Set T-curve
APS_set_axis_param(Master_Axis_ID, PRA_ACC,
     10000);// Set acceleration
APS_set_axis_param(Master_Axis_ID, PRA_DEC,
     10000);// Set deceleration
```

This way sets the master axis acceleration and deceleration to 10000 pulses per square of second during performing 2-axes interpolation motion. Above program is also set master axis to perform the velocity profile with T-curve.

```
…
Ret = APS_absolute_linear_move ( Dimension,
     Axis_ID_Array, Position_Array,
     Max_Linear_Speed );
…
```

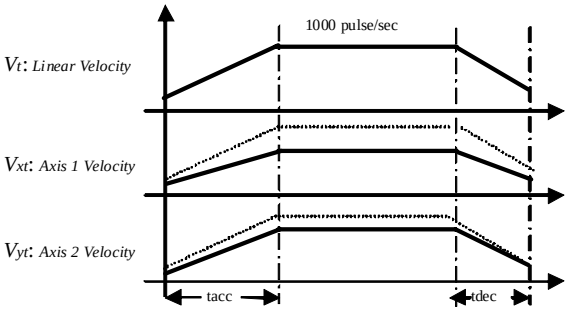
After whole parameter setting completed and then user is able to perform the linear interpolation function. If needs operating in relative mode, perform function **`APS_relative_linear_move`** (). The change in the velocity profile is illustrated in the diagram below.

For example, according to preceding description that start vector speed linear interpolation will be equal to the start velocity of master axis, is 10 pulses per second and its end velocity is also equal to the end velocity of master axis, is 30 pulses per second. In this case, PCI-8392 / PCI-8392H inner, DSP will handle the velocity profile of first and second axis to match two axes interpolation motion requirement. Basically, PCI-8392 / PCI-8392H calculates the acceleration time-'tacc' and deceleration time-'tdec' and then generates the actual acceleration/deceleration rate for axis 1 and 2 which based on the mathematical model (ratio) as following.

$$Vt = \sqrt{(Vxt)^2 + (Vyt)^2}$$

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

In which V_t is vector speed of linear interpolation motion; V_{xt} is speed along x direction (first axis); V_{yt} is speed along y direction (second axis).

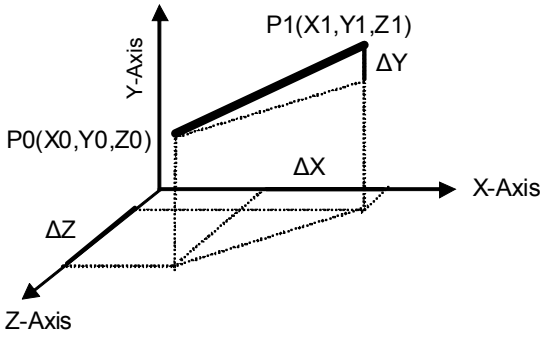


As illustration above, the x and y axis will be achieved the maximum speed at the same time that means the linear speed also achieves maximum linear speed in the meanwhile.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

3 Axes Linear Interpolation

Any 3 of the 16 axes of SSCNET board may perform 3 axes linear interpolation. As the figure below, 3 axes linear interpolation means to move the XYZ (if axes 0, 1, 2 are selected and assigned to be X, Y, Z respectively) position from P0 to P1 and start and stop simultaneously. The path is a straight line in space.



The speed ratio along X-axis, Y-axis and Z-axis is (X: Y: Z), respectively, and the vector speed is:

$$\frac{\Delta P}{\Delta t} = \sqrt{(\frac{\Delta X}{\Delta t})^2 + (\frac{\Delta Y}{\Delta t})^2 + (\frac{\Delta Z}{\Delta t})^2}$$

When calling the 2 axes linear interpolation functions, it is the **vector speed** to define the start velocity (master axis), **PRA_VS**, and maximum velocity, **Max_Linear_speed**, both trapezoidal and S-curve profile are available.

Example:

```
//…Initialize card
I32 Dimension = 4;
I32 Master_Axis_ID = 1; //Master axis
I32 Axis_ID_Array[4] = {1, 2, 3}; //Axis ID 1 is
     master axis.
I32 Position_Array [4] = {1000, 2000, 3000}; //
     (Unit: pulse)
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

This cause the four axes (axes 1, 2 and 3) to perform a linear interpolation movement, in which:

```
ΔX = 1000 pulse
ΔY = 2000 pulse
ΔZ = 3000 pulse

I32 Max_Linear_Speed = 1000;  //(Unit: pulse/
     second)
```

This way sets the maximum speed to 1000 pulse per second during performing 4-axes interpolation motion.

```
I32 Ret;
APS_set_axis_param(Master_Axis_ID, PRA_CURVE, 0);
     //Set T-curve
APS_set_axis_param(Master_Axis_ID, PRA_ACC,
     10000);// Set acceleration
APS_set_axis_param(Master_Axis_ID, PRA_DEC,
     10000);// Set deceleration
```

This way sets the master axis acceleration and deceleration to 10000 pulses per square of second during performing 3-axes interpolation motion. Above program is also set master axis to perform the velocity profile with T-curve.

```
…
Ret = APS_absolute_linear_move (Dimension,
     Axis_ID_Array, Position_Array,
     Max_Linear_Speed );
…
```

After whole parameter setting completed and then user is able to perform the linear interpolation function in absolute mode. The change in the velocity profile is illustrated in the diagram below.

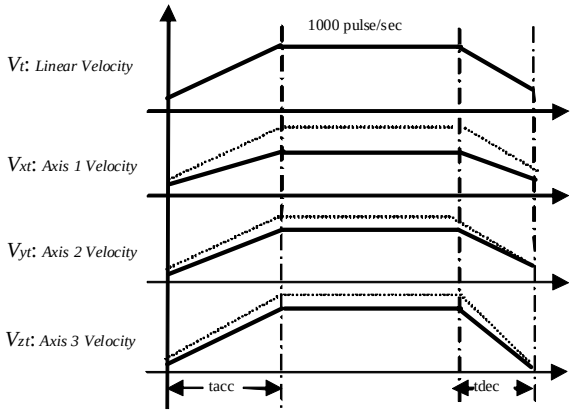
For example, according to preceding description that start vector speed linear interpolation will be equaled to start velocity of master axis, is 10 pulses per second and its end velocity is also equaled to end velocity of master axis, is 30 pulses per second. In this case, PCI-8392 / PCI-8392H inner, DSP will handle the velocity profile of first and second axis to match two axes interpolation motion requirement. Basically, PCI-8392 / PCI-8392H calculates the acceleration time-‘tacc’ and deceleration time-‘tdec’ and then

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

generates the actual acceleration/deceleration rate for axis 1 and 2 which based on the mathematical model (ratio) as following.

$$Vt = \sqrt{(Vxt)^2 + (Vyt)^2 + (Vzt)^2}$$

In which V_t is vector speed of linear interpolation motion; V_{xt} is speed along x direction (first axis); V_{yt} is speed along y direction (second axis); V_{zt} is speed along y direction (third axis).



As illustration above, the x and y axis will be achieved the maximum speed at the same time that means the linear speed also achieves maximum linear speed in the meanwhile.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4 Axes Linear Interpolation

In 4 axes linear interpolation, the speed ratio along X-axis, Y-axis, Z-axis and U-axis Is (ΔX: ΔY: ΔZ: ΔU), respectively, and the vector speed is:

$$\frac{\Delta P}{\Delta t} = \sqrt{(\frac{\Delta X}{\Delta t})^2 + (\frac{\Delta Y}{\Delta t})^2 + (\frac{\Delta Z}{\Delta t})^2 + (\frac{\Delta U}{\Delta t})^2}$$

Note: All axes must be of the same card.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.2.4 Circular Interpolation

Any 2 of the 16 axes of SSCNET board can perform circular interpolation. As the example below, the circular interpolation means XY (if axes 0, 1 are selected and assigned to be X, Y respectively) axes simultaneously start from initial point, (0,0) and stop at end point,(1800,600). The path between them is an arc, and the **`Max_Arc_Speed`** is the tangential speed

Example:

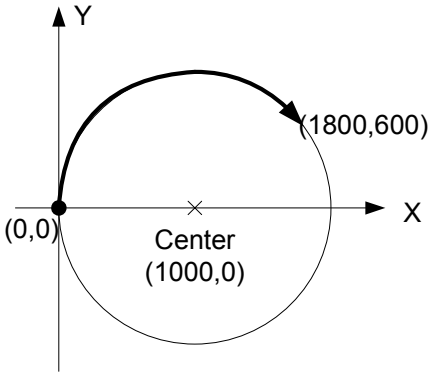
```
//…Initial card
I32 Dimension = 2;
     //2 Dimension only
I32 Axis_ID_Array[2] = { 2, 4 };                /
     /Axis_ID 2 is the master axis
I32 Master_Axis_ID = 2;                         /
     /Axis_ID 2 is the master axis
I32 Center_Pos_Array[2] = {1000, 0};    //Set
     center of circle(Unit: pulse)
I32 Max_Arc_Speed = 1000;               // pulse/
     sec
I32 Angle = -143;
     // clockwise 180 degree
I32 Ret;
     //Return code
```

To specify a circular interpolation path, the following parameters have to be defined clearly.

Center point: The coordinate of the center of arc (in absolute mode) or the offset distance to the center of arc (in relative mode)

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Angle: The moving angle, either clockwise (-) or counter clockwise (+)




```
//…
APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 1
     );  //Set S-curve
APS_set_axis_param( Master_Axis_ID, PRA_ACC,
     100000 ); //Set acceleration
APS_set_axis_param( Master_Axis_ID, PRA_DEC,
     100000 ); //Set deceleration
Ret = APS_absolute_arc_move( Dimension,
     Axis_ID_Array, Center_Pos_Array,
     Max_Arc_Speed, Angle ); //Perform a circular
     interpolation
```

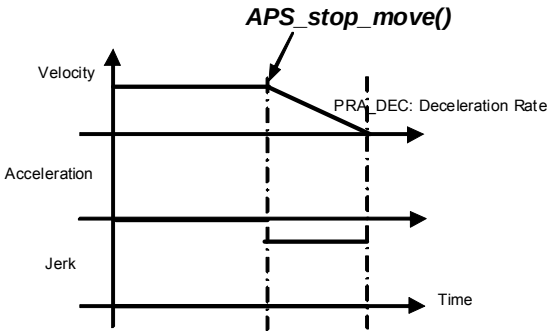
After whole parameter setting completed and then user is able to perform the circular interpolation function in absolute mode.

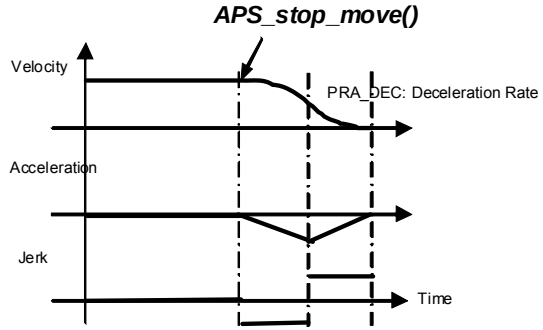
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.2.5 Speed Override

Speed override means that users can change command's speed during the operation of motion. User is easy to modify the speed profile during motion operation by re-executing the function **APS_velocity_move**().This function can be applied on other motion in position or velocity mode. If the running motion is S-curve or bell curve, the speed override will be a pure s-curve. If the running motion is t-curve, the speed override will be a t-curve.

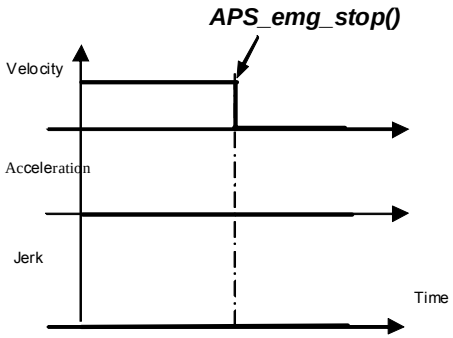
The function **APS_stop_move**() and **APS_emg_stop**() are used to stop a moving axis. If user wants to adjust the moving speed of an axis and just need to overwrite the velocity move function again. Function **APS_stop_move**() stops the specified '**Axis**' with a deceleration rate and a "Trapezoidal" or "S-Curve" velocity profile during deceleration. See diagram below.





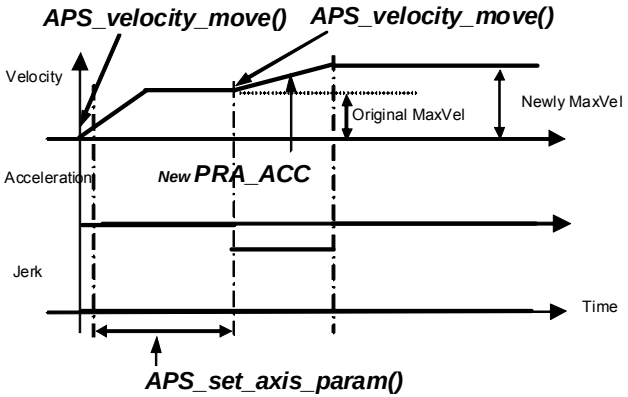
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

The function **APS_emg_stop**() stops the a specified 'Axis' immediately without deceleration. See diagram below.

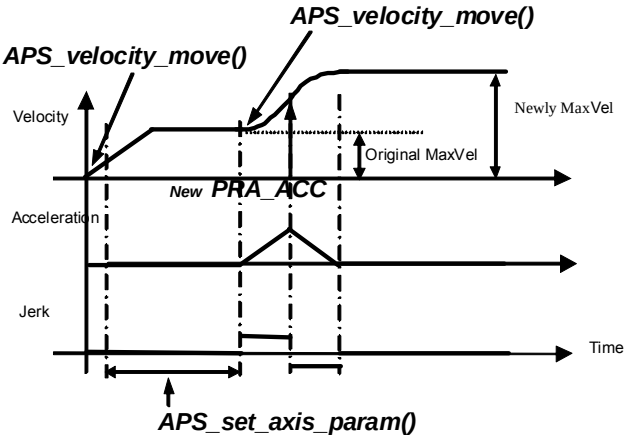


By re-perform the function **APS_velocity_move**(), user is able to changes the moving speed of a specified '**Axis**' with specified acceleration rate, '**PRA_ACC**', and a 'Trapezoidal' velocity profile during acceleration. If user needs different acceleration rate between original and newly parameter setting and then has to perform the function **APS_set_axis_param**() to re-define the newly velocity profile with newly acceleration rate.

The newly parameter table is available to fill during previous motion traveling.



LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



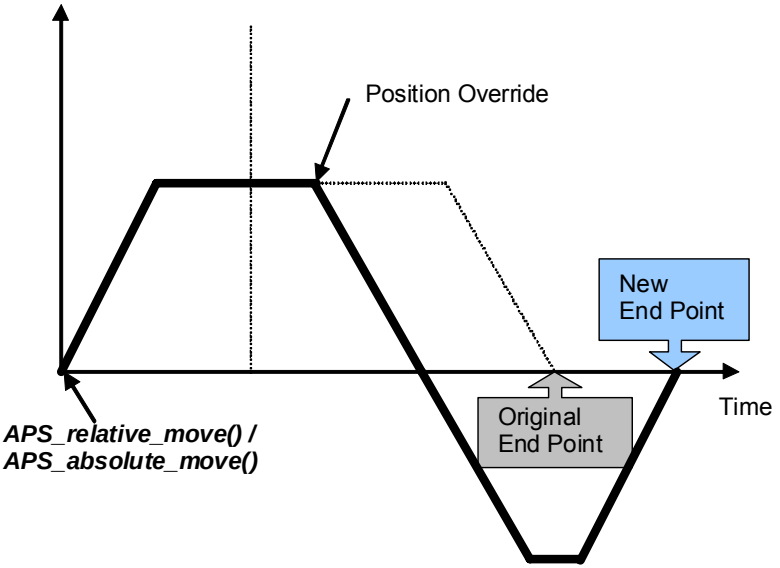
Use the same method to change the velocity during axis traveling with S-curve velocity profile.

Note: All change speed on the fly function calls can be applied any time when an axis is moving but cannot be overridden by other motion modes like Jog, home, manual pulse generation, contour motion and G-code program. The axis motion **must be stopped** before switching to those modes mentioned above.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.2.6 Position Override

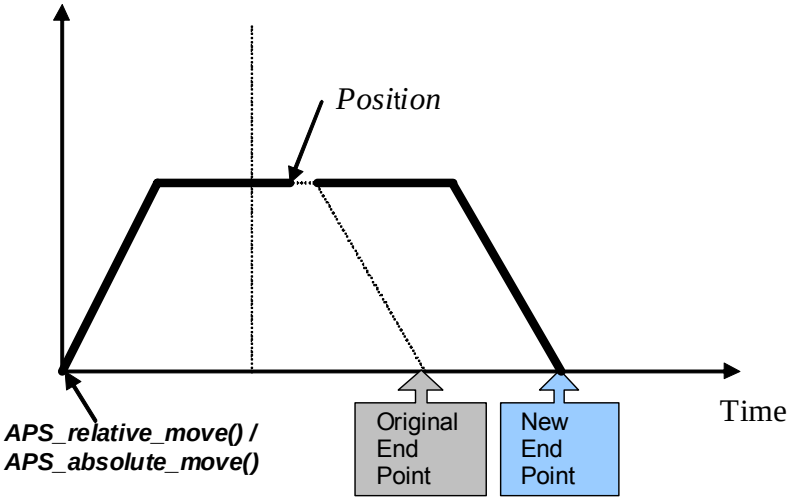
Position override means that when users issue a positioning command and want to change its target position during this operation. If the new target position is behind current position when override command is issued, the motor will slow down then reverse to new target position. If the new target position is far away from current position on the same direction, the motion will remain its speed and run to new target position. If the override timing is on the deceleration of current motion and the target position is far away from current position on the same direction, it will accelerate to original speed and run to new target position. The operation examples are shown as below. Notice that if the new target position's relative pulses are smaller than original slow down pulses, this function can't work properly.

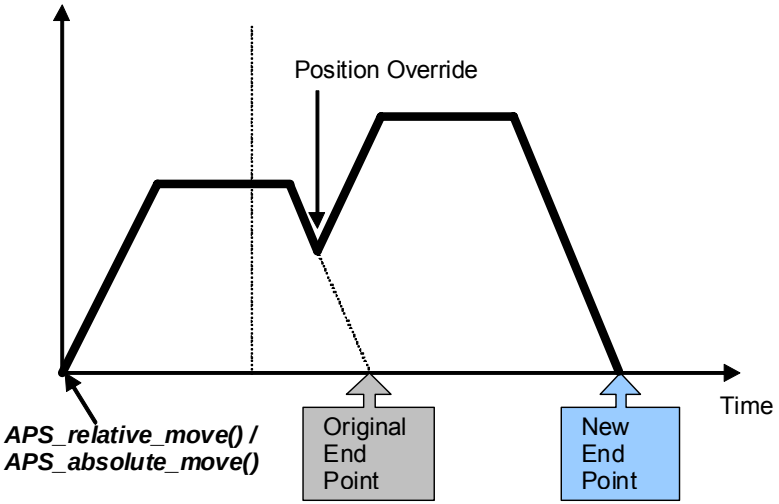


LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

By re-performing the function **APS_relative_move**() or **APS_absolute_move**(), you can easily able to changes the destination of a specified '**Axis**' with specified acceleration / deceleration rate, '**PRA_ACC**', '**PRA_DEC**' and a '**Trapezoidal**' velocity profile during acceleration. If you need different acceleration rate between original and newly parameter setting and then has to perform the function **APS_set_axis_param**() to re-define the newly velocity profile with newly acceleration / deceleration rate.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000





LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.3 Home Move

In this section, the following functions are discussed.

```
APS_home_move(Axis_ID)
```

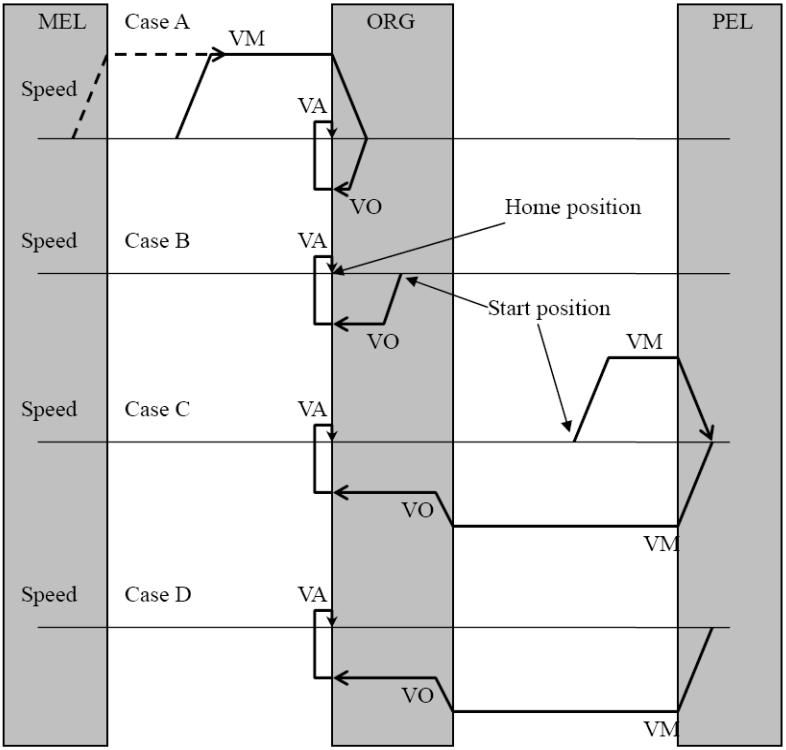
The above function is used to set the home return mode. One Home modes are available in PCI-8392 / PCI-8329H right now and it is discussed below.

After configured whole parameters that applied in homing move operation by function **APS_set_axis_param**(). User may use the **APS_home_move**() function to command the axis to start returning home. The '**PRA_HOME_VS**' defines the starting velocity for homing move, '**PRA_HOME_VM**' defines the end velocity for homing move, the '**PRA_HOME_ACC**' define the acceleration rate and the axis continues traveling at the constant velocity until it reaches the ORG switch. Furthermore, in PCI-8392 /PCI-8392H, there defines the parameter '**PRA_HOME_VO**' that means the leaving velocity after achieved home (ORG / Zero position). You are able to use this parameter to accelerate the motion speed on hoimg procedure. Home move examples using only the ORG signal are illustrated as follows:

PRA_HOME_MODE	0	Homemode 1
PRA_HOME_DIR	0	Positive direction
PRA_HOME_CURVE	0	T-curve
PRA_HOME_EZC	0	Disable
PRA_HOME_ACC	ACC	Acceleration / decelaration
PRA_HOME_VS	0 (VS)	Start velocity
PRA_HOME_VM	VM	Max. velocity
PRA_HOME_VO	VO	ORG velocity

Table 4-1: Axis Parameter Settings for ORG Home Moves

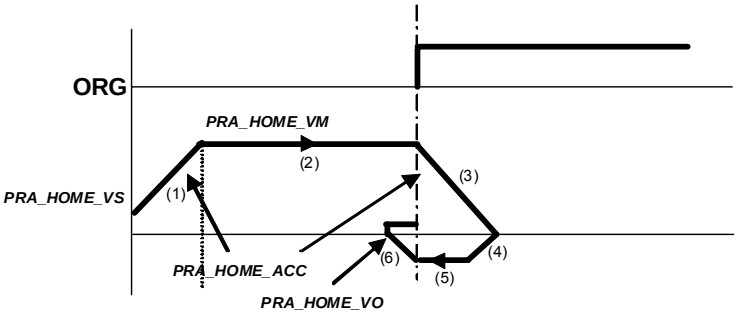
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



VM : Home move max. speed
VO : Home move ORG speed
VA : Home move approach to home speed

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example



1. Accelerate from '**PRA_HOME_VS**' to '**PRA_HOME_VM**'.
2. Travel with constant velocity '**PRA_HOME_VM**' until ORG turns ON.
3. Slow done to stop which is following '**PRA_HOME_ACC**'.
4. Return and accelerate to '**PRA_HOME_VO**'.
5. Travel with constant velocity '**PRA_HOME_VO**' until ORG turn Off.
6. Slow down to stop.

Searching ORG rising edge with velocity = 1 pulse per SSCNet cycle time until ORG turn ON, then stop and finish home return.

Example:

```
//Set homing parameters
APS_set_axis_param( Axis_ID, PRA_HOME_MODE, 0 );
      //Set home mode

APS_set_axis_param( Axis_ID, PRA_HOME_DIR, 1 );
      //Set home direction

APS_set_axis_param( Axis_ID, PRA_HOME_CURVE, 0 );
      //Set acceleration paten (T-curve)

APS_set_axis_param( Axis_ID, PRA_HOME_ACC,
      1000000 ); //Set homing acceleration rate
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
APS_set_axis_param( Axis_ID, PRA_HOME_VS, 0 );
     //Set homing start velocity

APS_set_axis_param( Axis_ID, PRA_HOME_VM, 2000000
     ); //Set homing maximum velocity.

APS_set_axis_param( Axis_ID, PRA_HOME_VO, 200000
     ); //Set leaving home velocity

APS_home_move(Axis_ID ); //Start homing

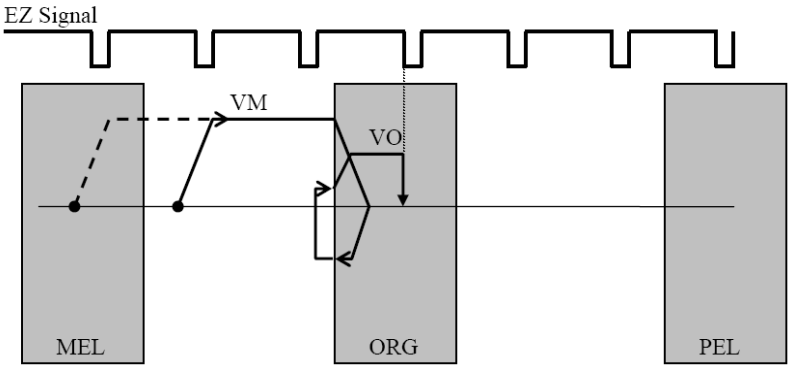
…//Check homing done(Motion done)
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

In addition to home moves using the ORG signal, this board also offers home moves via the index signal (EZ) as illustrated:

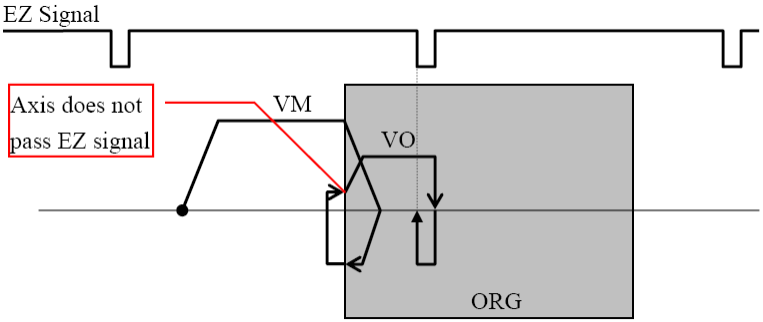
PRA_HOME_MODE	0	Homemode 1
PRA_HOME_DIR	0	Positive direction
PRA_HOME_CURVE	0	T-curve
PRA_HOME_EZC	**1**	**Enable**
PRA_HOME_ACC	ACC	Acceleration / decelaration
PRA_HOME_VS	0 (VS)	Start velocity
PRA_HOME_VM	VM	Max. velocity
PRA_HOME_VO	VO	ORG velocity

Table 4-2: Axis Parameter Settings for EZ Home Moves



Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

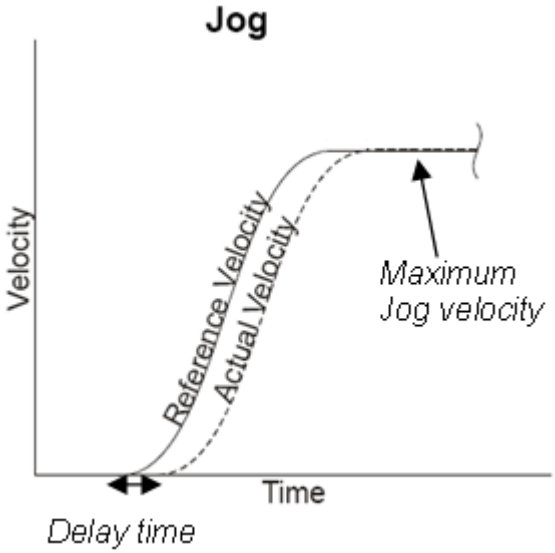
This board offers additional home move functions for specified situations. If the axis does not pass the EZ signal before the axis approaches the ORG signal and detectes the ORG signal, the axis will continuous move to detect the EZ signal. After passing the EZ signal, the axis will stop and move back to the edge of the EZ signal.



Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.4 Jogging

Jog motion usually uses only a velocity command. There is no fixed target position, only a target velocity. There is two modes for jogging motion which one is ‘Free-running mode’ and another is ‘Step mode’. ‘Free-running mode’ means the jog motion applies the motion with only velocity command. ‘Step mode’ means the jog motion is performing with specified step command. All on-the-fly velocity change comments in the point-to-point profile apply here.



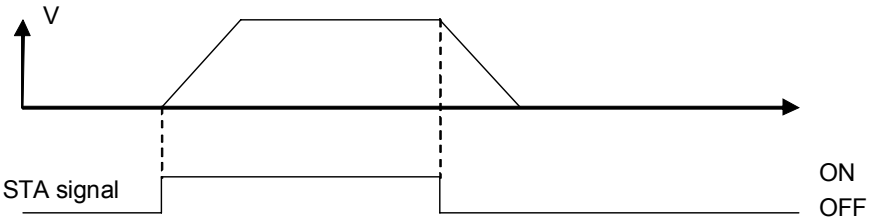
Free-running Mode:

The jog mode of motion is very flexible because speed, direction and acceleration can be changed during motion. For jog motion operation, the user is able to specifie the maximum jog speed, acceleration, and the deceleration rate for each axis by function ‘**APS_set_jog_param**()’. The direction of motion is specified by the sign of the jog parameters which be set with function ‘**APS_set_jog_param**()’. When the begin command is given ‘**APS_jog_start**()’, the motor accelerates up to speed and continues to jog at that speed until a new speed or stop command

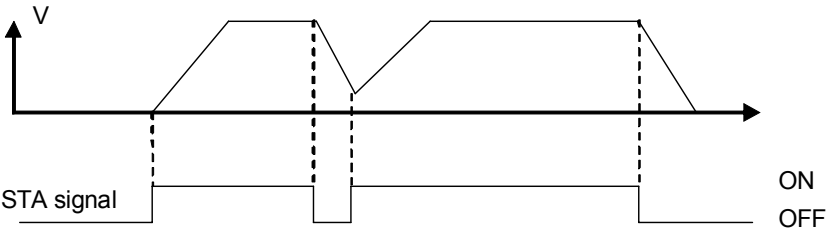
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

'**APS_stop_move**()' is issued. If user wants to change the jog speed during motion, the above function will be needed to set again to meet new jog motion command. The Jog move is illustrated as follows,

When the STA signal is turn on, the axis starts to move immediately. Then, when the STA signal turns off, the axis will start decelerating until it stops.



When the axis is decelerating and the STA signal is turned on again, the axis will re-accelerate to the maximum velocity.

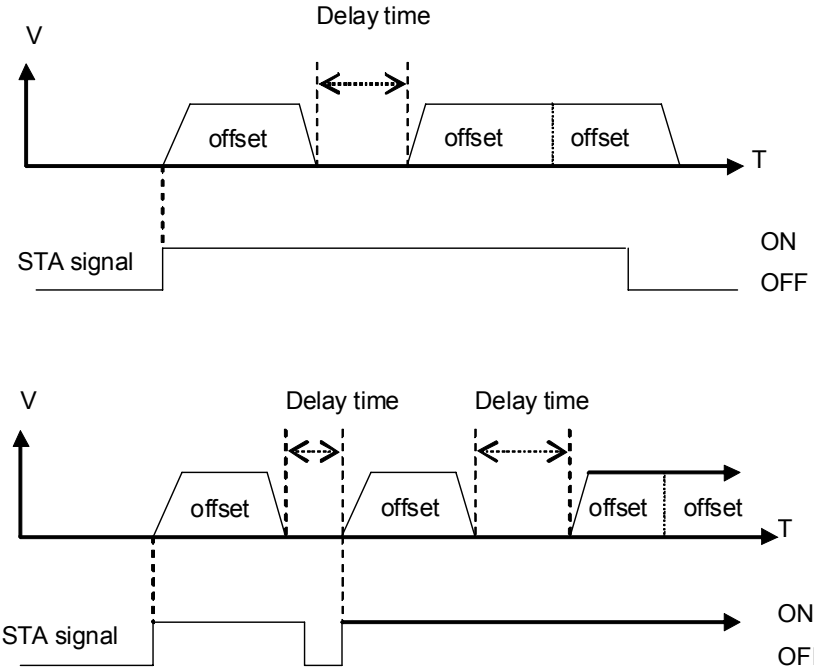


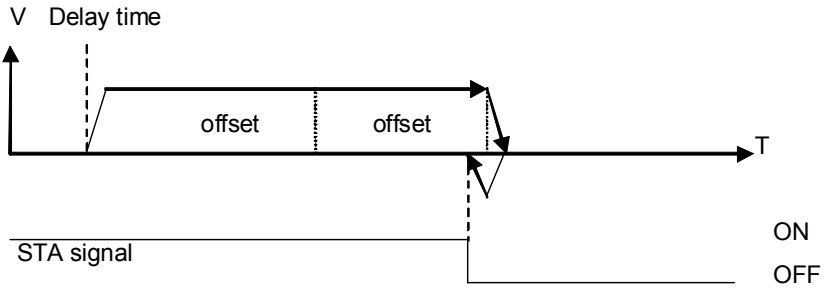
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Step Mode:

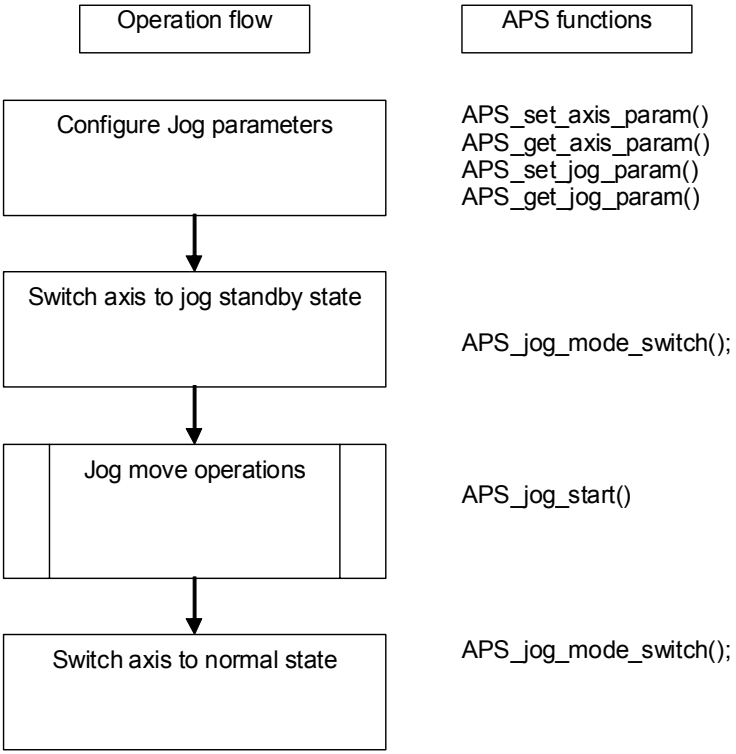
An instant change to the motor position can be made with the use of the step move command. Through jog mode setting by function '**APS_set_jog_param**()', Upon receiving this command, the controller commands the motor to a position which is equal to the specified increment plus the current position. User must set the step offset firstly and then perform the jog switch function '**APS_jog_mode_switch**()' to enable the axis to jog mode. Finally, performing start function '**APS_jog_start** ()'. In this mode, user is also able to set the delay time for jog motion.

As shown in the following figure, when the STA signal is turned on, the axis starts accelerating and moves a specific distance (offset) before stopping. After a time delay, if the STA signal is still on, the axis will start to move untill the STA signal is turned off. The biggest different between free run mode is that the jog distance of an axis is the multiple number of the specified offset. Therefore, you can easily control the movement of an axis.





Below is the programming flow for jogging move.



LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Before executing the jog motion, user has to config whole parameters of jog motion that included the jog mode, moving direction, acceleration and deceleration rate and so on. An example is as follows.

Example:

```
//…Initial card
// … Configure jog move parameter

I32 Axis_ID: The Axis ID from 0 to 65535.
JOG_DATA *pStr_Jog: Structure of jog move
     parameters. Define in "type_def.h"
typedef struct
{
     I16 i16_jogMode; //Jog mode. 0:Free running
     mode, 1:Step mode
     I16 i16_dir;  //Jog direction. 0:positive,
     1:negative direction
     I16 i16_accType; //Acceleration and
     Deceleration pattern 0: T-curve, 1: S-curve
     I32 i32_acc;  //Acceleration rate (pulse /
     s2)
     I32 i32_dec; //Deceleration rate (pulse /
     s2)
     I32 i32_maxSpeed; //A Positive value,
     maximum velocity.(pulse/s)
     I32 i32_offset;  //A Positive value, step
     offset. (For step jog Mode ONLY. (pulse))
     I32 i32_delayTime; // Delay time (For step
     jog mode ONLY) ( range: 0 ~ 65535
     millisecond, align by cycle time)
} JOG_DATA;
```

After configed the jog parameters and then user has to perform the switch function to enable this axis into jog mode. Noted the axis is ONLY operating in jog mode if user turned the jog mode on.

```
ret = APS_jog_mode_switch(Axis_ID, 1 );          /
     /Turn on jog move mode
…
// perform jog move …(APS_jog_start)
…
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

The axis will be actually performed the jog motion when controller receiving the start function '**APS_jog_start**()'. If other operations are selected to be performed after jog motion and the axis has to release from jog mode by performing the switch function again to disable this axis from jog mode.

```
APS_jog_start( Axis_ID,1 ); //STA signal ON
…
APS_jog_start(Axis_ID, 0); //STA signal OFF
…
ret = APS_jog_mode_switch(Axis_ID, 0 ); //Turn
      off jog move mode
…
// perform other move commands
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.5 Point Table

In this section, the operation of continuous motion is introduced and implemented with a point table. Continuous motion means a series of motion commands or positions that can be run continuously. You can set a new command right after previous one without interrupting it.

4.5.1 Point Table Construction

In this section, the following functions are discussed.

```
APS_set_point_table(Axis_ID, Index, POINT_DATA
     *Point)
APS_get_point_table(Axis_ID, Index, POINT_DATA
     *Point)
```

The **APS_set_point_table**() function is used to declare the parameters for the specified axis motion list describing a continuous motion trajectory. After the declaration for **APS_set_point_table**(), you can call the functions discussed in next section to piece-wisely extend the trajectory by adding 'Index'. In the meantime, each axis has a maximum of 32K points which can be saved in the onboard memory. By utilizing a double-buffered design, the point table size is able to regarded as infinite. After the upper 16 points are proceeded and finished the lower 16 points will then be proceeded continuously. While the lower 16 points is operating, the new points are able to be filled into the empty region (upper 16 points). In the same manner, when next upper 16 points is proceeded, the region of the previous lower 16 points will be filled. The maximum time delay of point update does not exceed 1 ms.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

The last parameter '**POINT_DATA *Point**' defines the total parameters of axes that will be involved in the continuous motion. The structure of '**POINT_DATA**' defines in "**type_def.h**" and listed in following table,

Parameter	Description	Unit / Range
_pos	Position data / Center (Arc trajectory)	pulse
_accType	Acceleration pattern	0: T curve ; 1: S curve
_decType	Deceleration pattern	0: T curve ; 1: S curve
_acc	Acceleration rate	pulse /
_dec	Deceleration rate	pulse /
_initSpeed	Start velocity	pulse / s
_maxSpeed	Maximum velocity	pulse / s
_endSpeed	End velocity	pulse / s
_angle	Arc move angle	-360 to 360 degrees
_dwell	Dwell times	ms
_opt	Point move option	(Refer to below table)

Table 4-3: POINT_DATA Structure

You are able to config the moving method, stop condition and moving type by parameter '**_opt**' definition and it is listed as follows,

Point move option: _opt

7	6	5	4	3	2	1	Bit : 0
-	-	Last point	Finish Condition	-	Linear/ Arc	-	Absolute/ Relative

Bit 0: 1: Relative move, 0: Absolute move

Bit 2: 1: Arc move, 0: Linear move

Bit 4: 1: INP ON (In position signal), 0: CSTP ON (command stop signal)

Bit 5: 1: Last point index. 0: Not the last point index (if this bit is turned on, the point table movement will stop after this point.)

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Bit ‘0’ decides the moving method to relative mode or absolute mode. Bit ‘2’ decides the trajectory moving following linear interpolation or circular interpolation method. Bit ‘0’ decides the moving method to relative mode or absolute mode. Bit ‘4’ decides the stop method that refers to ‘INP’ signal or ‘CSTP’ signal.

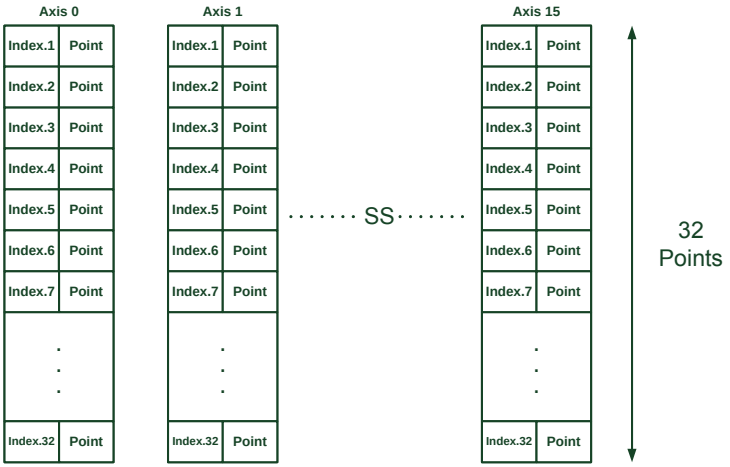
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.5.2 Point index

In order to easy to establish the point table, there uses point index '**Index**' to program or monitor the point sequence whether work well or not. In this section, the following functions are discussed.

```
APS_get_start_point_index(Axis_ID,* Index)
APS_get_running_point_index(Axis_ID,* Index)
APS_get_end_point_index(Axis_ID,* Index)
```

By using the **APS_get_xxx_point_index**() function, the point index is dedicated to identify the start point, current point, or end point which is operating in point table sequence.



The PCI-8392 and PCI-8392H allows point tables to be performed by different start point indexes to different end point indexes. In other word, the movement duration is programmable for different start points to end points. For example, if the depth of the established point table is 100 points and then the movement can be started from point index '5' to index '15'.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.5.3 Point table execution

In this section, the following function is discussed.

```
APS_point_table_move(Dimension, *Axis_ID_Array,
     StartIndex, EndIndex)
APS_set_table_move_pause(Axis_ID, Pause_en);
APS_set_table_move_repeat(Axis_ID, Repeat_en);
```

By using the **APS_xxx_table_move**() function, the point index is dedicated to identify the start point, next point, or end point which is operating in point table sequence. By using the aforementioned functions to pause the point table movement or set the point table movement to repeat, when the pause command is issued, it will not stop the current point but stop at the next point index. Furthermore, when the repeat function is enabled, it will repeat the point move until the repeat function is disabled or a stop function is issued.

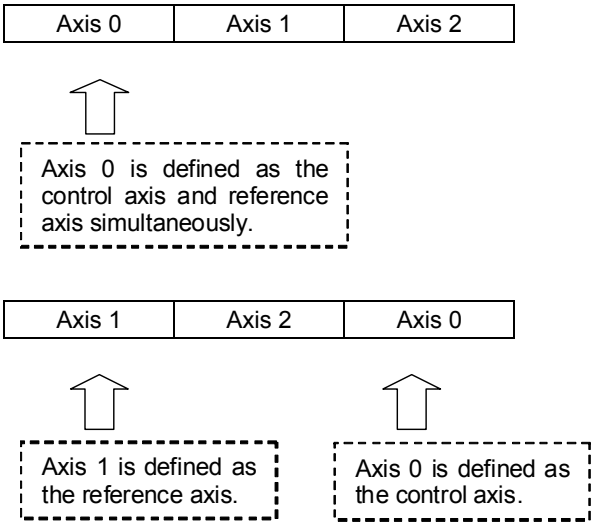
In order to carries out point table movement, the movement dimension, active axis number and its index must be defined first. Furthermore, the point table sector executed also needs to be defined before point table movement begins. '**Dimension**' means the maximum dimension which the point table is executed. If there is only one 3D coordinated point in the point table sequence, which the most points exists in 2D or 1D coordinate, the dimension of this point table must be set completely to 3D coordinate. If interpolation movememt is needed, the '**Axis_ID_Array**' has to set by your application. Other important properties of '**Axis_ID_Array**' also define the reference axis, control axis and slave axis while this axis array is set in interpolation movement. Refer to the following table. if axis 0 to axis is the active axes in your application, the allocation of the axis that will define the axis property to reference as control or slave.

Typically, the minimum axis number will be the control axis, meaning it decides the behavior of interpolation. Therefore, the slave axis will follow the control axis to execute its movement correspondingly. It allows for several existing slave axes. For the reference axis, if it is define to the first element of '**Axis_ID_Array**' and its parameter is able to be reused in other axes, in other word, if all the axes have the same move parameter in the motion opera-

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

tion, then you just need to program the related motion parameter for the reference axis.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000



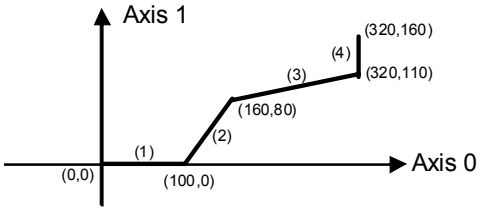
For another method, the point table execution is able to begin from any start point with '**StartIndex**' and then close with '**EndIndex**'.

There are several examples presented as follows:

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example: 2-D Linear move

The resulting 2-D trajectory is:




```
#include "type_def.h"
#include "APS_define.h"
#include "APS168.h"
#include "ErrorCodeDef.h"

I32 ret;
POINT_DATA Point_AX0, Point_AX1;
I32 Axis_ID_Array[2] = {0, 1};

// Set first position for AX0 and it could be
     relative or absolute value (Unit: Pulse)
Point_AX0.i32_pos = 100;
Point_AX0.i16_accType = 1;//Acceleration pattern
     0: T curve, 1:S curve
…
// Set finish condition to INP signal occurred ;
     set linear interpolation ; set absolute
     motion
Point_AX0.i32_opt = 9

//Set point data to card memory
ret = APS_set_point_table(Axis_ID[0], 0, &
     Point_AX0);
…
//Set and save parameter for AX1
…
// ********* Set remaining point *********

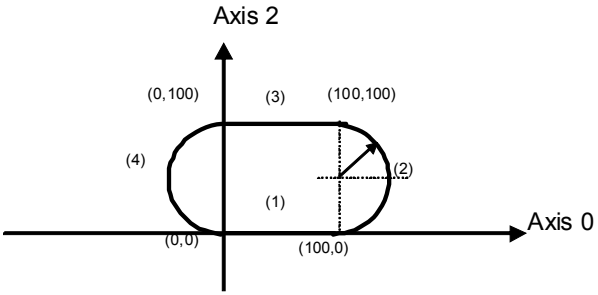
// Start a point table move
ret = APS_point_table_move( 2, Axis_ID_Array, 0 ,
     3 );
…
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

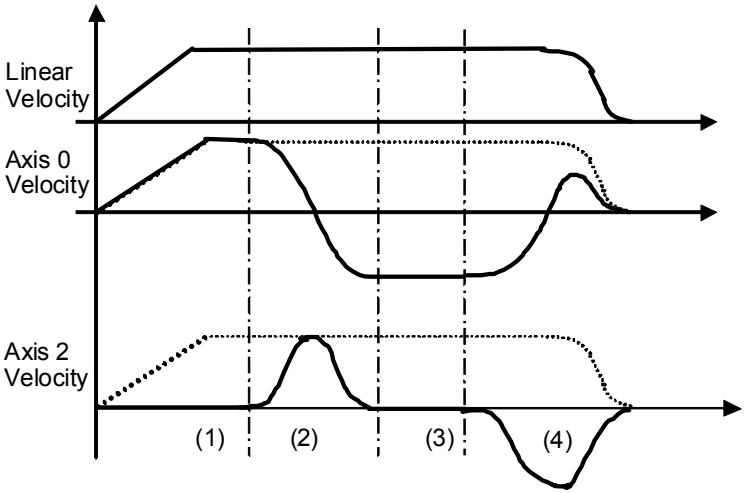
A 2-D arc trajectory is also able to add to the continuous motion trajectory in the point table.

Example: 2-D Linear Move

The resulting 2-D trajectory is:



The resulting velocity vs. time is:




```
#include "type_def.h"
#include "APS_define.h"
#include "APS168.h"
#include "ErrorCodeDef.h"
```

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

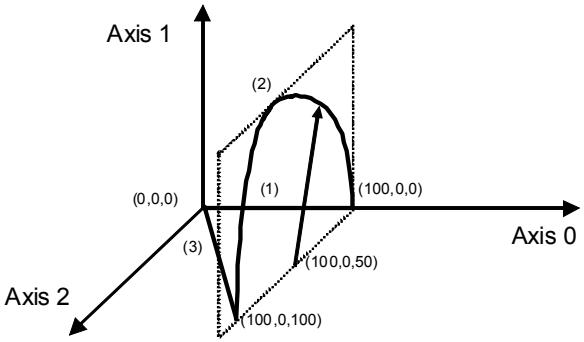
```
I32 ret;
POINT_DATA Point_AX0, Point_AX2;
I32 Axis_ID_Array[2] = {0, 2};
// *******First motion*******
// Set first position for AX0 (Unit: Pulse)
Point_AX0.i32_pos = 100;
Point_AX0.i16_accType = 1;//Acceleration pattern
     0: T curve, 1:S curve
…
// Set finished condition to INP signal occurred
     ; set linear interpolation ; set absolute
     motion
Point_AX0.i32_opt = 9
//Set point data to card memory
ret = APS_set_point_table(Axis_ID[0], 0, &
     Point_AX0);
…
//Set and save parameter for AX2
// *******Second motion*******
// Set center position for AX0 and it could be
     relative or absolute value (Unit: Pulse)
Point_AX0.i32_pos = 100;
…
Point_Ax0.i32_angle = 180 //(Unit: Degree)
Point_AX0.i32_opt = 13 //Set to circular
     interpolation
//Set point data to card memory
ret = APS_set_point_table(Axis_ID[0], 1, &
     Point_AX0);
…
// Set center position for AX2
Point_AX2.i32_pos = 50;
…
//Set point data to card memory
ret = APS_set_point_table(Axis_ID[1], 1, &
     Point_AX2);
…
//********* Set and save parameter for third and
     fourth motion ********
// Start a point table move
ret = APS_point_table_move( 2, Axis_ID_Array, 0 ,
     3 );
…
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

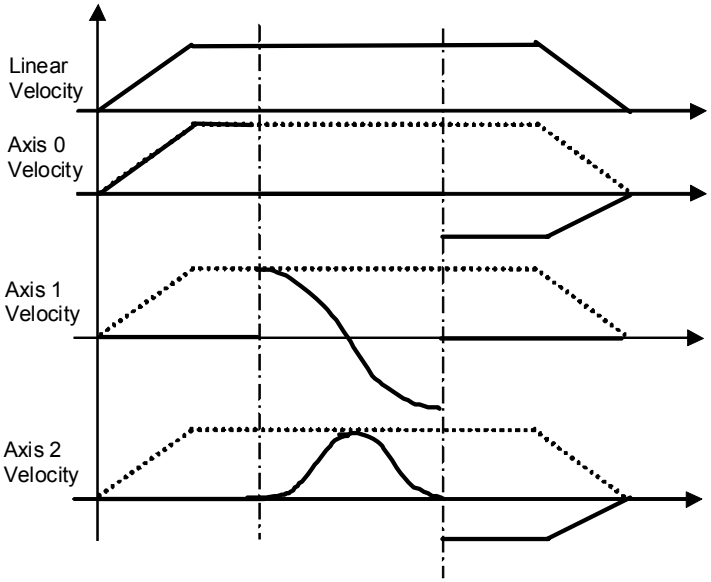
A 3-D trajectory is also able to add to the continuous motion trajectory in the point table.

Example: 2-D Linear Move

The resulting 3-D trajectory is:



The resulting velocity vs. time is:



LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
I32 ret;
POINT_DATA Point_AX0, Point_AX1, Point_AX2;
I32 Axis_ID_Array[3] = {0, 1, 2};

// ********First motion *******
// Set first position for AX0 (Unit: Pulse)
Point_AX0.i32_pos = 100;
Point_AX0.i16_accType = 1;//Acceleration pattern
     0: T curve, 1:S curve
…
// Set finished condition to INP signal occurred
     ; set linear interpolation ; set to absolute
     motion
Point_AX0.i32_opt = 9
//Set point data to card memory
ret = APS_set_point_table(Axis_ID[0], 0, &
     Point_AX0);
…
//Set and save parameter for AX1 and AX2

// *******Second motion *******

// Set center position for AX0
Point_AX0.i32_pos = 100;
…
Point_Ax0.i32_angle = 180 //(Unit: Degree)
Point_AX0.i32_opt = 13 //Set to circular
     interpolation ; set to absolute motion
//Set point data to card memory
ret = APS_set_point_table(Axis_ID[0], 1, &
     Point_AX0);
…
// Set center position for AX1
Point_AX1.i32_pos = 0;
…
Point_Ax1.i32_angle = 180
Point_AX1.i32_opt = 13
…
ret = APS_set_point_table(Axis_ID[1], 1, &
     Point_AX1);
…
// Set center position for AX2
Point_AX2.i32_pos = 50;
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
…
Point_Ax2.i32_angle = 180
Point_AX2.i32_opt = 13
…
ret = APS_set_point_table(Axis_ID[2], 1, &
      Point_AX2);

//********* Set and save parameter for third
      motion ********************

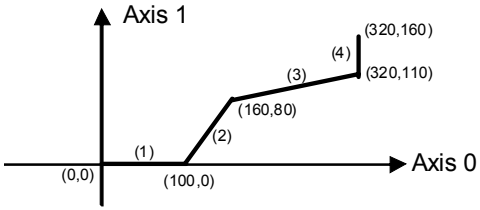

// Start a point table move
ret = APS_point_table_move( 3, Axis_ID_Array, 0 ,
      2 );
…
```

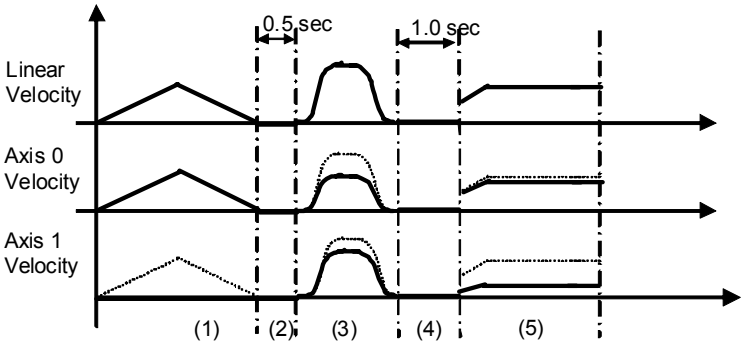
When the motion operation is smooth, the motion can also be frozen for a specified period of time by inserting '**Dwell**', definied by parameter 'dwell' in units of millisecond. If the dwell time was set in point table sequence, the index will be occupied. For example, as in the 2-D linear movement above, the motion procedure is configured into four steps so that the point table index 0 to index 3 are used. However, if you decided to insert two dwell times between the first motion, second motion and third motion accordingly, the total index used will be from index 0 to index 5 because there are two indexs be used to insert dwell time. The following is an example:

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example: Insert Dwell into Motion Sequence

The resulting velocity vs. time is:






```
I32 ret;
POINT_DATA Point_AX0, Point_AX1;
I32 Axis_ID_Array[2] = {0, 1};

// Set first position for AX0 and it could be
     relative or absolute value (Unit: Pulse)
Point_AX0.i32_pos = 100;
Point_AX0.i16_accType = 1;//Acceleration pattern
     0: T curve, 1:S curve
…
Point_AX0.u32_dwell = 500 ; //Set dwell time to
     0.5 s
// Set finish condition to INP signal occurred ;
     set linear interpolation ; set absolute
     motion
Point_AX0.i32_opt = 9
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
//Set point data to card memory
ret = APS_set_point_table(Axis_ID[0], 0, &
      Point_AX0);
…
//***Using index 1 to insert first dwell time ***

Point_AX0.u32_dwell = 500 ; //Set dwell time to
      0.5 s

//Set dwell data to card memory
ret = APS_set_point_table(Axis_ID[0], 1, &
      Point_AX0);

//Set and save parameter for AX1
…
//***Set next point***
…
//***Using index 3 to insert first dwell time***

Point_AX0.u32_dwell = 1000 ; //Set dwell time to
      1 s

//Set dwell data to card memory
ret = APS_set_point_table(Axis_ID[0], 3, &
      Point_AX0);

//***Set remaining point***


// Start a point table move
ret = APS_point_table_move( 2, Axis_ID_Array, 0 ,
      5 );
…
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.6 Motion Status and Related IO Monitoring

In addition to the SSCNet servo motor control capabilities, the SSCNET board has other I/O functions and can roughly be divided into 2 categories. They are the motion related I/O's and the motion status. Motion related I/O's are input and output signals dedicated to motion. For example: PEL/MEL, position/velocity feedback…etc. Furthermore, the motion status will be real-timed update by motion status monitoring function. This section will concentrate on the motion related I/O and their function calls.

- Section 4.6.1: Position Control and Feedback
- Section 4.6.2: Motion I/O Status
- Section 4.6.3: Motion Status

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.6.1 Position Control and Feedback

In this section, the following functions are discussed.

```
APS_set_position(Axis_ID, Position)
APS_get_position(Axis_ID, *Position)
APS_set_command(Axis_ID, Command)
APS_get_command(Axis_ID, *Command)
```

Set position

The **APS_set_position**() function allows users to set a current position counter value for the servo driver.

Get position information

The SSCNET board controls servo drivers & motors via an SSC-Net protocol. For each SSCNet cycle (0.888ms), the SSCNET board sends a command to and receives a response from the servo driver. Through command and response, an abundant amount of information is carried in and out, including position command and position feedback. The function call **APS_get_position**() will retrieve such information. The parameter '*Position' retrieves the current position feedback, which is reported from the servo driver through SSCNet communication.

Set command

This function is used to set the position information of one axis. The information is in unit of pulse. The function '**APS_set_command**()' will change current position or command to a new one. Notice that the set command may cause axis to move violently in closed loop control or absolute control system. However, it is no problem in open loop system. The set position command is legal in closed loop or absolute control system because it will also set command to the same value at the same time.

Get command information

For each SSCNet cycle (0.888ms), the SSCNET board sends a command to and receives a response from the servo driver. The function call **APS_get_command**() will retrieve command information which be calculated by DSP and the parameter '*Command' retrieves the next command, user is easy to monitor the command status and verify the generated profile by such function.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.6.2 Motion I/O Status

In this section, the following function is discussed.

```
APS_motion_io_status(Axis_ID)
```

The “motion DIO” mentioned here refers to the motions dedicated to the digital I/O signals including PEL, MEL, ORG, and EMG. Each axis has its own motion DIO signal except EMG. All axes from a single card shares the same EMG signal. User has to config the related I/O logic by function ‘**APS_set_axis_param**()’ firstly to insure that the signal works legally. In other way, for detailed of motion I/O status, user is able to refer to motion I/O status table to check each I/O bit definition in function library section.

End-limit signals

The end-limit signals are used to stop the axis when they are active. There are two possible stop modes, one is “stop immediately”, and, the other is “decelerate to start velocity then stop”. The parameter ‘**PRA_EL_Mode**’ in ‘**APS_set_axis_param**()’ are used to select the mode. Furthermore, user is able to use either an ‘a’ contact switch or a ‘b’ contact switch by setting the parameter ‘**Logic**’.

PEL signal indicates the end-limit in the positive (plus) direction. The MEL signal indicates the end-limit in the negative (minus) direction. When the axis is moving towards the positive direction, the axis will be stopped when the PEL signal becomes active, while the MEL signal is no affect in this case, and vise versa. When the PEL is active, only the negative (minus) direction motion is allowed.

The PEL/MEL signals can generate an IRQ, if the interrupt service routine is enabled. Refer to section 4.10.

The PEL/MEL status can be monitored through the function **APS_motion_io_status**().

ORG signal

The ORG signal is used, when the axis is operating under the home return mode. The logic polarity of the ORG signal is selectable using the parameter ‘**Logic**’ of ‘**APS_set_axis_param**()’. The ORG status can be monitored using the function **APS_motion_io_status**().

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

EMG signal

Each SSCNET board has an EMG signal input. Whenever this EMG signal becomes active, all the axes control by in the card will stop moving immediately.

The EMG signal is capable of generating an IRQ if an interrupt service routine is enabled, refer to section 4.10.

The logic polarity of the EMG signal is selectable using the parameter '**Logic**' of '**APS_set_axis_param**()'. The EMG status can be monitored using the function **APS_motion_io_status**().

Except for the above mentioned motion I/O, this function retrieves more motion related I/O status simultaneously which included alarm signal, in position signal, servo on signal and so on. For detailed information of motion I/O status, user can refer to motion I/O table to understand the definition of each bit as following table.

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
SVON	INP	EZ	EMG	ORG	MEL	PEL	ALM
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
	ABSL	TLC	SMEL	SPEL	ZERO	WARN	RDY

Table 4-4: Motion IO Status Bit Definition

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.6.3 Motion Status

In this section, the following function is discussed.

```
APS_motion_status(Axis_ID)
```

This function is used to get one axis' motion status. The status includes running, normal stop, abnormal stop by reasons, in waiting other axis, follow status, in some modes, in accelerating or decelerating and so on. Status can be more than two such like mode and running. Users need to use this function to check whether the 'Fire-and-forget' function is done in polling system. In even driven system, users can use interrupt event functions. For detailed of each bit of motion status, user is able to refer to motion status table to check each bit definition in function library section.

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
SMV	HMV	NSTP	DIR	DEC	ACC	VM	CSTP
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
JOG	SLV	PPS	PDW	PMV	SMO	CIP	LIP
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
SEMGS	MELS	PELS	WANS	ALMS	EMGS	SVONS	ASTP
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**
--	ERRS	--	--	--	STPOA	SMELS	SPELS

Table 4-5: Motion status definition table

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.7 Driver Management

In this section, all servo driver related functions are discussed. Including:

- Section 4.7.1:Driver parameter
- Section 4.7.2:Servo On
- Section 4.7.3:Servo Alarm

4.7.1 Driver parameter

In this section, the following functions are discussed.

```
APS_get_sscnet_servo_param(Axis_ID, Para_No1,
     *Para_Dat1, Para_No2, *Para_Dat2)
APS_set_sscnet_servo_param(Axis_ID, Para_No1,
     Para_Dat1, Para_No2, Para_Dat2)
APS_save_sscnet_servo_param(Board_ID)
```

With the SSCNET board, servo parameters read/write becomes very easy using function calls listed above. There is different method to access the servo parameter between previous designed sscnet board, which provided by ADLINK, and PCI-8392/PCI-8392H. In the PCI-8392/PCI-8392H, user is able to access two parameter packages once by using sscnet protocol. For example, user can set '**Para_No1**' to PA10 and '**Para_Dat1**' from 0 - 5000 to set the in-position range, moreover, user is also able to set '**Para_No2**' to PA14 and '**Para_Dat2**' for 0 or 1 to decide the rotation direction in the meanwhile. Like as above mentioned, user is also able to read two parameter packages in the same time.

To read a current parameter setting, user can call **APS_get_sscnet_servo_param**(). This function retrieves certain parameter values settings.

To set a new value for the servo parameters, user can call **APS_set_sscnet_servo_param**(). This function will set new values for specified parameters only.

After a servo parameter tuning process, user may use **APS_save_sscnet_servo_param**() to store current parameter setting. These values are stored in the Flash ROM which on the SSCNET board.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

For more information for Mitsubishi J3B parameter details that is ble to refer to the “MR-J3B Instruction Manual”.

4.7.2 Servo On

In this section, the following function is discussed.

```
APS_set_servo_on(Axis_ID, Servo_on)
```

This function is able to command one axis to servo on individually, moreover, if user wants to monitor the status for servo on/off, the function **APS_motion_status**() is useful to indicate the axis’s servo current status.

4.7.3 Servo Alarm

In this section, the following functions are discussed.

```
APS_get_sscnet_servo_alarm(Axis_ID,*Alarm_No,
     *Alram_Detail)
APS_reset_sscnet_servo_alarm(Axis_ID)
```

When a fault occurs, the servo driver will stop the motor and report an alarm number on the LED display of the servo driver. The SSCNET board will also be acknowledged because the servo driver also reports the alarm condition through SSCNet communication. If this is the case, users can exam the fault condition by:

1. Examining the return code from the function call.
2. Set an IRQ. Allow a IRQ to be generated, if an alarm occurs. Refer to Section 4.10, page 91.

After noticing the occurrence of the alarm, users can use **APS_get_sscnet_servo_alarm**() to retrieve the alarm code, which is displayed on the LED of the servo driver. After removing the alarm condition, users can use the **APS_reset_sscnet_servo_alarm**() function to recover the driver from the alarm state.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.8 Data Sampling

In this section, the following functions are discussed.

```
APS_set_sampling_param(Board_ID, Param_No,
     Param_Dat)
APS_get_sampling_param(Board_ID, Param_No,
     *Param_Dat)
APS_wait_trigger_sampling(Board_ID, Length,
     PreTrgLen, TimeOutMs, STR_SAMP_DATA_4CH
     *DataArr)
```

In order to help user to easy analyze the motion related profile or data. Data sampling conception was addressed to users and it is able to regard as digital scope. Similar to oscilloscope, in PCI-8392 / PCI-8392H, there gives each axis 4 monitoring channels, trgger level setting, trigger source selection function ans so on. Users can use these monitoring channels to monitor a variety of I/O data, such as feedback velocity, INP (in position) signal status, etc.

To be able to use the monitoring function, users must understand the configuring and operating procedures.

Configuring procedures:

Configuring procedure is necessary before a monitor function startes. There are two main instructions during configuration:

1. Set_monitor_config

   This function is used to set the monitoring configuration, such as sampling rate, trigger condition…etc. This function must be executed before monitoring starts. The function **`APS_set_sampling_param`**() is used to config the related parameter and set the action triggered channel. The first parameter 'Param_No' specifies the sampling parameter numbers that are listed below.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Sampling parameters definition table

Para NO.	Define	Description	Parameter data value.
00h	SAMP_PA_RATE	Sampling rate(cycle) (depending on cycle time)	1- 65535 (times of cycle)
02h	SAMP_PA_EDGE	Edge triggered	0:Rising edge 1:faling edge
03h	SAMP_PA_LEVEL	Triggered level	(I32) -2147483648 to 2147483647
05h	SAMP_PA_TRIGCH	Trigger channel	0 - 3 (Ch0-Ch3)
10h	SAMP_PA_SRC_CH0	Sampling source of Channel 0	Refer to sampling source table
11h	SAMP_PA_SRC_CH1	Sampling source of Channel 1	Refer to sampling source table
12h	SAMP_PA_SRC_CH2	Sampling source of Channel 2	Refer to sampling source table
13h	SAMP_PA_SRC_CH3	Sampling source of Channel 3	Refer to sampling source table

Table 4-6: Sampling Parameters Definition Table

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2. Set_monitor_channel

This function is used to set the monitoring target. This function must be executed before monitoring can started. The function **`APS_set_sampling_param`**() is also used to set the monitoring channel. The first parameter '**`Param_No`**' specifies the sampling parameter numbers that are listed below. The remaining parameters '**`Param_Dat`**' are used for the monitoring targets. For general usage, each channel is available to config to same trigger source with individual axis, or config to different trigger source with same axis, the relationship was illustrated as following table.

Source	Symbol Define	Description	Value Range
00h	SAMP_COM_POS	Command position (pulse)	I32 value
01h	SAMP_FBK_POS	Feedback position (pulse)	I32 value
02h	SAMP_CMD_VEL	Command velocity (pps)	I32 value
03h	SAMP_FBK_VEL	Feedback velocity (pps)	I32 value
04h	SAMP_MIO	Motion IO status (Same as Get motion IO function)	I32 value (bit format)
05h	SAMP_MSTS	Motion status (Same as Get motion status function)	I32 value (bit format)
06h	SAMP_MSTS_ACC	Motion status at acceleration (Command velocity)	0: Not at acceleration 1: At acceleration
07h	SAMP_MSTS_MV	Motion status at max velocity (Command velocity)	0: Not at max. velocity 1: At max. velocity
08h	SAMP_MSTS_DEC	Motion status at deceleration (Command velocity)	0: Not at deceleration 1: At deceleration
09h	SAMP_MSTS_CSTP	Motion status command stop (CSTP)	0: CSTP status ON 1: CSTP status OFF

Table 4-7: Sampling Source Definition Table

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Source	Symbol Define	Description	Value Range
0Ah	SAMP_MSTS_NSTP	Motion status normal stop (NSTP)	0: NSTP status ON 1: NSTP status OFF
0Bh	SAMP_MIO_INP	Motion status in position (INP)	0: INP status ON 1: INP status OFF
0Ch	SAMP_MIO_ZERO	Motion status zero (ZERO)	0: ZERO status ON 1: ZERO status OFF
0Dh	SAMP_MIO_ORG	Motion status ORG status	0: OGR status ON 1: OGR status OFF

Table 4-7: Sampling Source Definition Table

Operating procedures:

Like as the above statement, the data sampling conception is similar to oscilloscope. There are 2 major operation procedures which have to execute.

1. Configure trigger condition

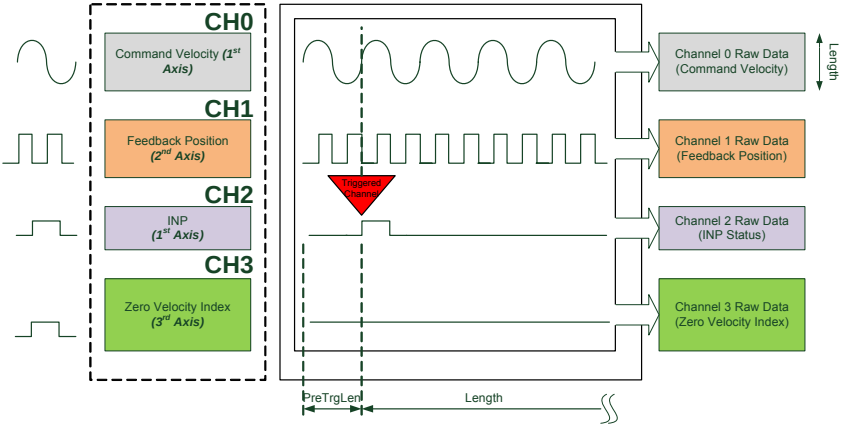
   Before the sampling starts, user has to conig related triggred condition that included triggered sampling rate, triggered edge selection, triggered level. After the triggered condition completion, user has to set trigger source which user wants to monitor with determined channel.

   This function returns a value immediately and carries out real time monitoring of specified monitoring targets.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2. Select the sampling source

After sampling condition configuration completion, user needs to decide what kinds date wants to monitor, in PCI-8392 / PCI-8392H, ADLINK offers 14 sampling sources for monitoring and they are listed as the above table.



Example:

```
//... initialize card
APS_set_sampling_param( Board_ID, SAMP_PA_RATE, 2
     ); //Set sampling rate
APS_set_sampling_param( Board_ID, SAMP_PA_EDGE, 0
     ); //Set trigger edge (rising edge)
APS_set_sampling_param( Board_ID, SAMP_PA_LEVEL,
     1  ); //Set trigger level (1)
APS_set_sampling_param( Board_ID, SAMP_PA_TRIGCH,
     2  ); //Set trigger channel (channel 2)
```

The sampling condition needs to be configured in advance. In this example, the sampling rate was set to two multiple of cycle time and defines the triggered edge to rising type. Furthermore, the triggered level was set to ‘1’ because the I/O status just has defined to either ‘high’ or ‘low’. Finally, the channel was picked as triggered channel.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
APS_set_sampling_param( Board_ID,
     SAMP_PA_SRC_CH0, SAMP_CMD_VEL ); //Set
     channel_0 sampling source
APS_set_sampling_param( Board_ID,
     SAMP_PA_SRC_CH1, SAMP_FBK_POS  ); //Set
     channel_1 sampling source.
APS_set_sampling_param( Board_ID,
     SAMP_PA_SRC_CH2, SAMP_MIO_INP ); //Set
     channel_2 sampling source
APS_set_sampling_param( Board_ID,
     SAMP_PA_SRC_CH3, SAMP_MIO_ZERO  ); //Set
     channel_3 sampling source.
```

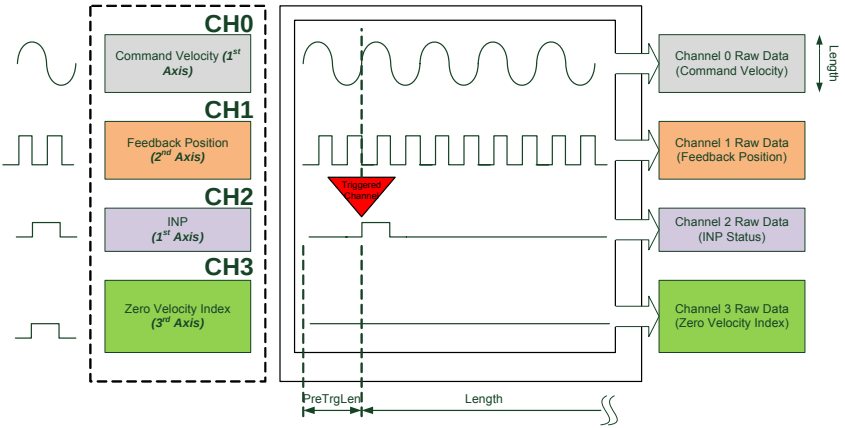
After sampling condition configuration completion, user still has to select the triggered source for each sampling channel.

```
I32 Length = 1024; //Total sampling data array
     size
I32 PreTrgLen = 100; //The number of pre-trigger
     points
STR_SAMP_DATA_4CH DataArr[1024];
I32 TimeOutMs = 10000; //10 second timeout

Ret =APS_wait_trigger_sampling( Board_ID, Length,
     PreTrgLen, TimeOutMs, DataArr );
//…
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Finally, you need to define the storage length for sampling raw data. After the function **APS_wait_trigger_sampling**() was executed, the 1024 raw data was stored in the DataArr. Other advantage was also offered in this function, user is able to set the parameter '**PreTrgLen**' and it defines the length of pre-buffer, like as oscilloscope, user is also able to monitor the raw data before triggered event occurred.



Example:

```
//... initialize card
APS_set_sampling_param( Board_ID, SAMP_PA_RATE, 2
     ); //Set sampling rate
APS_set_sampling_param( Board_ID, SAMP_PA_EDGE, 0
     ); //Set trigger edge (rising edge)
APS_set_sampling_param( Board_ID, SAMP_PA_LEVEL,
     1  ); //Set trigger level (1)
APS_set_sampling_param( Board_ID, SAMP_PA_TRIGCH,
     2  ); //Set trigger channel (channel 2)
```

Similar to previous example, you have to config the sampling condition in advance. The related condition was configed as above setting.

```
APS_set_sampling_param( Board_ID,
     SAMP_PA_SRC_CH0, SAMP_CMD_VEL ); //Set
     channel_0 sampling source
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
APS_set_sampling_param( Board_ID,
     SAMP_PA_SRC_CH1, SAMP_FBK_POS  ); //Set
     channel_1 sampling source.
APS_set_sampling_param( Board_ID,
     SAMP_PA_SRC_CH2, SAMP_MIO_INP ); //Set
     channel_2 sampling source
APS_set_sampling_param( Board_ID,
     SAMP_PA_SRC_CH3, SAMP_MIO_ZERO  ); //Set
     channel_3 sampling source.
```

After sampling condition configuration completion, user still has to select the triggered source for each sampling channel.

```
I32 Length = 1024; //Total sampling data array
     size
I32 PreTrgLen = 100; //The number of pre-trigger
     points
STR_SAMP_DATA_4CH DataArr[1024];
I32 TimeOutMs = 10000; //10 second timeout

Ret =APS_wait_trigger_sampling( Board_ID, Length,
     PreTrgLen, TimeOutMs, DataArr );

//…
```

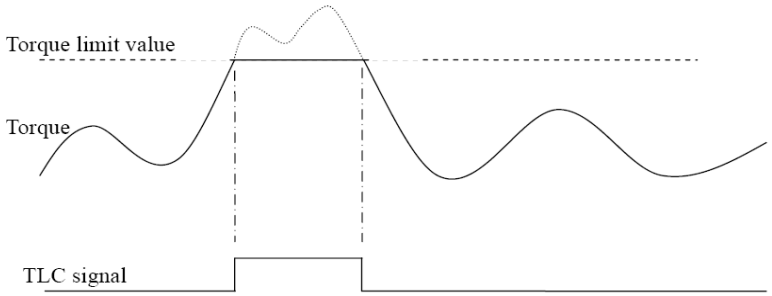
Finally, you need to define the storage length for sampling raw data. After the function APS_wait_trigger_sampling() was executed, the 1024 raw data was stored in the DataArr. Other advantage was also offered in this function, user is able to set the parameter ‘PreTrgLen’ and it defines the length of pre-buffer, like as oscilloscope, user is also able to monitor the raw data before triggered event occurred.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.9 Torque Limit

When you enabled the torque-limited control function, the torque of servo driver will be limited under the specified “torque-limit value”.

When torque-limited control function is turned on and torque is over the torque limit value, the motion IO status “TLC” will turned on.



LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Relative functions:

- **APS_set_axis_param** () Set axis parameter
- **APS_get_axis_param** () Get axis parameter
- **APS_motion_io_status**() Return motion IO status

Axis parameters definitions are as follows:

NO.	Definition	Description	Value	Note
0001 0004h	PRA_SSC_ TORQUE_ LIMIT_P	Positive torque limit value (0.1%)	0~32767 (Default 3000)	(*2)
0001 0005h	PRA_SSC_ TORQUE_ LIMIT_N	Negative torque limit value (0.1%)	0~32767 (Default 3000)	(*2)
0001 0006h	PRA_SSC_ TORQUE_ CTRL	Torque control enable	0: Disable, (Default, Control with motor max. torque) 1: Enable, (Control with torque limit value)	(*1)

Table 4-8: SSCNET Axis Parameters Definition Table

Note: (*1) Some SSCNET axis parameters will be rest to default value when you start SSCNET network.

(*2) 0.1% Set 1000 mean 100%

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example

```
#include “APS168.h”
#include “APSDefine.h”

//Start SSCNET network
//Set torque-limit value

I32 torqueLimitValue_p = 1000;
I32 torqueLimitValue_n = 2000;
APS_set_axis_param( Axis_ID,
     PRA_SSC_TORQUE_LIMIT_P, torqueLimitValue_p
     );
APS_set_axis_param( Axis_ID,
     PRA_SSC_TORQUE_LIMIT_P, torqueLimitValue_n
     );


//Enable torque-limit control function

APS_set_axis_param( Axis_ID, PRA_SSC_TORQUE_CTRL,
     1 );


//Enable Polling MIO status

I32 motionIo;
motionIo = APS_motion_io_status(Axis_ID );
if( motionIo >= 0 )
{
if( (motionIo >> 13) & 1 )
{
… //Torque-limit control is turned on
}
else
… //Does not over torque-limit value
}
}
//…
```

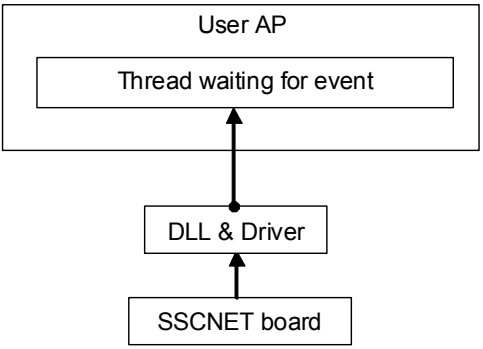
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.10 Interrupt Control

In this section, the following functions are discussed.

```
APS_int_enable(Board_ID, Enable)
ASP_set_int_factor(Board_ID, Item_No, Factor_No,
     Enable)
ASP_get_int_factor(Board_ID, Item_No, Factor_No,
     *Enable)
APS_wait_single_int(Int_No, Time_Out)
APS_wait_multiple_int(Int_Count, *Int_No_Array,
     Wait_All, Time_Out)
APS_reset_int(Int_No)
APS_set_int(Int_No)
```

The SSCNET board can generate an interrupt for certain conditions. Refer to the figure below:



Users can either set a call back routine that will be executed when an interrupt occurs, or create a thread to wait for an event that will be triggered when an interrupt occurs.

To enable or disable the interrupt generated from the SSCNET board, use the **APS_int_enable**() function. It acts as an **ON_OFF** switch. Once disabled, the SSCNET board will cease to generate any interrupt signals to the host system.

In addition to **APS_int_enable**(), users need to define the conditions under which an interrupt signal should occurs by using the **ASP_set_int_factor**() function in order to successfully introduce an interrupt signal to the host system. The parameter '**Item_No**' of **APS_set_int_factor**() is use to specify the axis

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

source with '**Factor_No**' specifying the interrupt factor for this specified source.

When the user enabled the interrupt function for specified factors by **ASP_set_int_factor**(), it could use this function to wait a specific interrupt by **APS_wait_single_int**(). When this function was running, the process would never stop until the event was be triggered or the function was time out. This function returns when one of the following occurs:

1. The specified interrupt factor is in the signaled state.
2. The time-out interval elapses.

This function checks the current state of the specified interrupt factor. If the state is non-signaled, the calling thread enters the wait state. It uses no processor time while waiting for the INT state to become signaled or the time-out interval to elapse.

Except single interrupt triggering function, this board also offers the multiple interrupt event triggering function **APS_wait_multiple_int**(), the most different points are the function will return the event triggered signal when one of the following occurs:

1. Either any one or all of the interrupt factors are in the signaled state.
2. The time-out interval elapses.

No matter what interrupt method selected, when the interrupt was occurred and the wait function is return. User should use **APS_reset_int** () to reset the interrupt by themselves. If user does not reset the interrupt, the wait function will pass immediately next time.

Item	Description
0	Axis 0 interrupt factors
1	Axis 1 interrupt factors
…	…
15	Axis 15 interrupt factors
16	System interrupt factors

Table 4-9: Interrupt Factor Table

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Item 0~15: Axes interrupt factors number definition

7	6	5	4	3	2	1	0
IZERO	IWARN	IINP	IEZ	IORG	IMEL	IPEL	IALM
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
--	ITLC	IASTP	INSTP	IDEC	IACC	IVM	ICSTP

Axes interrupt factors condition description

NO.	Define	Interrupt condition description
0	IALM	Servo alarm signal turn ON
1	IPEL	Positive end limit switch is turn ON
2	IMEL	Minus (Negative) end limit switch turn ON
3	IORG	Home switch turn ON
4	IEZ / IEZP	EZ passed signal turn ON
5	IINP	In position signal turn ON
6	IWARN	Servo warning ON
7	IZERO	Zero speed
8	ICSTP	Command stop
9	IVM	In maximum velocity
10	IACC	In acceleration
11	IDEC	In deceleration
12	INSTP	Normal stop(Motion done)
13	IASTP	Abnormal stop
14	ITLC	Torque limit control is turn ON
15	Reserved	--

1. In a SSCNET system, when zero position signal (EZ) from servo driver is ON, **EZP** bit will ON even if EZ is turn OFF.
2. **INSTP**: Axis is stopped normally. If axis is stopped abnormally such as emergency stop and Limit switch on stop etc, this interrupt factor will not be triggered. All motion action including home move which can be waited motion done by this interrupt factor.
3. You can set normal stop (motion done) condition by set axis parameter function.
4. **CSTP**: Motion command stopped, but the axis could be still in motion.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.11 Absolute Position System

In this section, the following function is discussed.

```
APS_get_sscnet_servo_abs_position(Axis_ID,
     *Cyc_Cnt, *Res_Cnt)
APS_save_sscnet_servo_abs_position(Board_ID)
APS_load_sscnet_servo_abs_position(Axis_ID,
     Option, *Cyc_Cnt, *Res_Cnt)
```

SSCNET board provides absolute position system for SSCNET motor driver. Users need only execute homing procedure once then the board can keep the absolute position in the ROM. When the machine restarts next time, it will restore the absolute position of each axis from the ROM. The machine need not to do the home procedure again.

Mitsubishi servo drivers use a battery to keep its encoder's value inside at absolute position mode. The position value is on servo driver and users must turn on absolute position mode in parameter "AMS" before using this feature. Of course, users need to install a battery to keep the position value on driver's side permanently.

In order to have absolute position feature on SSCNET motion control card, we must read the absolute position value from servo driver and retrieve the origin position information from FLASH ROM of SSCNET motion control card when card is initialized. After that we will calculate an absolute position of users' machine according to these two information.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

The formal procedure to use this features are as followings:

- Launch **APS_home_move**() function to complete homing.
- Check if the home position is correct
- Launch **APS_save_sscnet_servo_abs_position**()) to store the ABS position as an origin position reference.
- Next time, when SSCNET motion control card starts, you don't need to launch **APS_home_move**() anymore. They can only launch **APS_get_sscnet_servo_abs_position**() function to get an absolute position.

Remember to set this ABS position to operating position counter via **APS_set_position**() function. Don't use **APS_get_sscnet_servo_abs_position**() in polling cycle because it is much slower than **APS_get_position**() function.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.12 E-Gear

This function is used to adjust the command pulse number of one revolution (Resolution) for a servo system.

Users must set the “User define resolution” before start SSCNET network.

$$\text{Move ratio (R)} = \frac{\text{Encoder resolution}}{\text{User - define resolution}(2^N)}$$

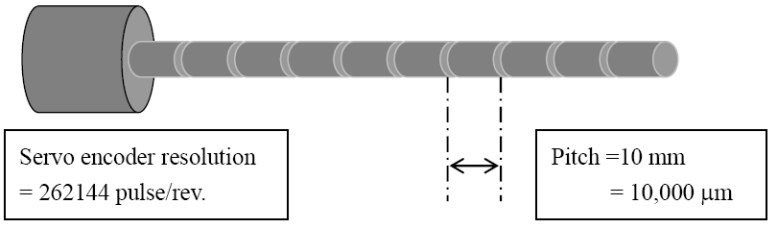
User-defined resolution = 2N; N= 12 ~ 18

(Axis parameter: **PRA_SSC_RESOLUTION**)

For Mitsubishi MR-J3-B servo motor, its encoder resolution is 262144 pulse / rev. The following table shows the relationship between parameter N, user-define resolution and move ratio R.

N	User define res. Pulse / Rev.	E-gear ratio (R)
18	$2^N = 262144$	1
17	$2^N = 131072$	2
16	$2^N = 65536$	4
15	$2^N = 32768$	8
14	$2^N = 16384$	16
13	$2^N = 8192$	32
12	$2^N = 4096$	64

Example:



When N = 13, (R = 32)

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

$$\text{Linear resolution} = \frac{\text{Screw pitch}}{\text{Encoder resolution}} \times \frac{\text{Encoder resolution}}{\text{User define res.}} (R)$$

$$= \frac{10\,\text{mm}}{262144} \times 32 \quad = 0.001221\ \text{mm/pulse}$$

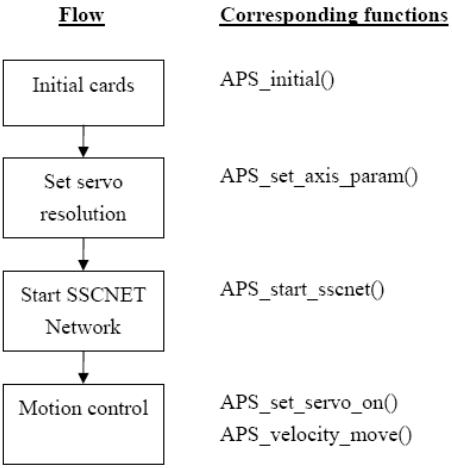
$$= \frac{10{,}000\,\mu\text{m}}{262144} \times 32 \quad = 1.221\ \mu\text{m/pulse}$$

Distance: 1,000 pulse = 1,000 pulse X 1.221 μm/pulse = 1221 μm

Velocity: 1,000 pps = 1,000 pulse/sec X 1.221 μm/pulse

= 1221 μm/sec.

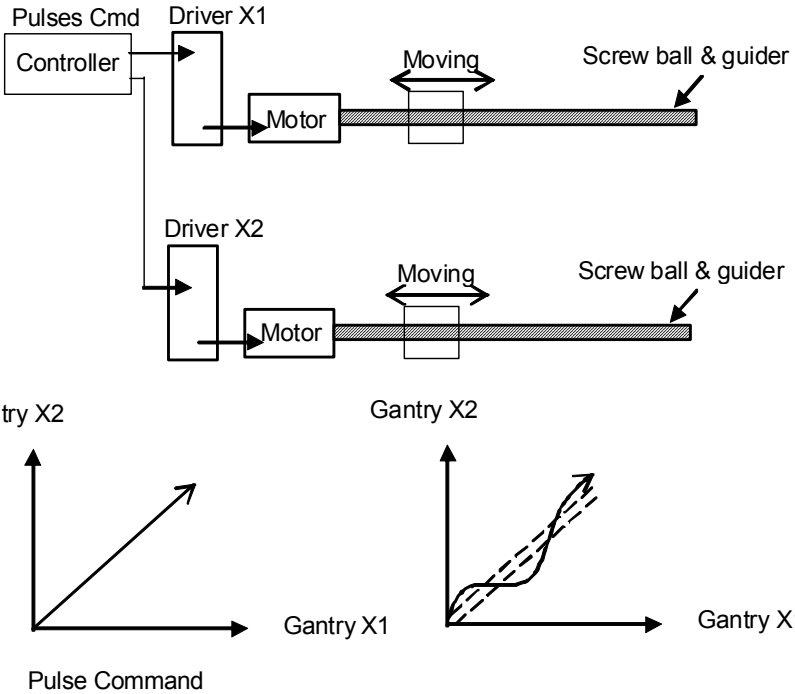
Programming flow example :



LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.13 DSP-based Gantry Function

Gantry function is defind from mechanical structure. Normally we have two motors controlling two parallel linear stages. These two motors must move similutaneously. The motion controller must control these two motors individually but synchronously. It looks like 1:1 electric gearing in master-slave motion and also looks like 1:1 interpolation motion of two axes. If using ASIC motion, you can use linear motion function of two axes to achieve this function at PC side. It is only for command pulse synchronization. Some applications, especially on unbalanced heavy loading conveyor system, it doesn't work perfectly because of open loop architecture. Please see the following diagram:

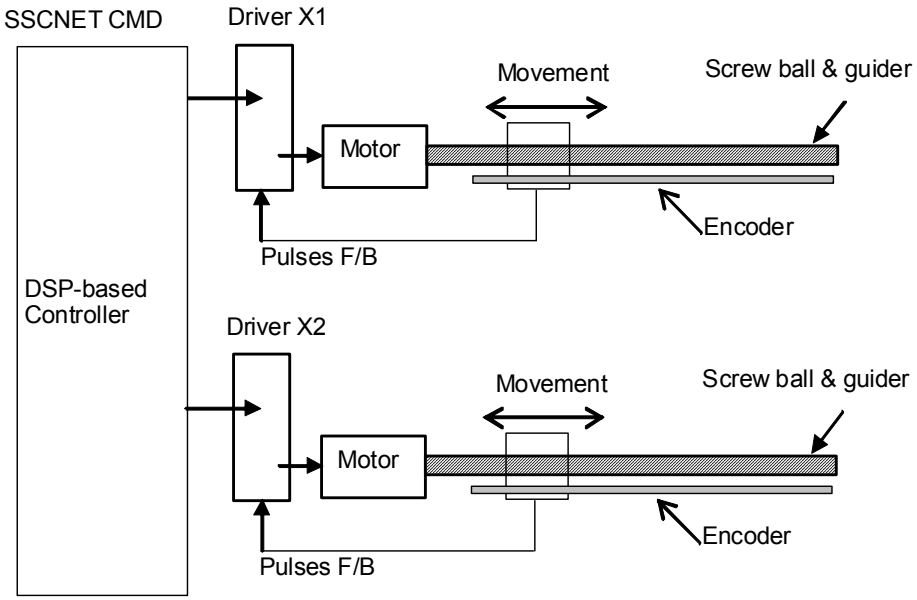


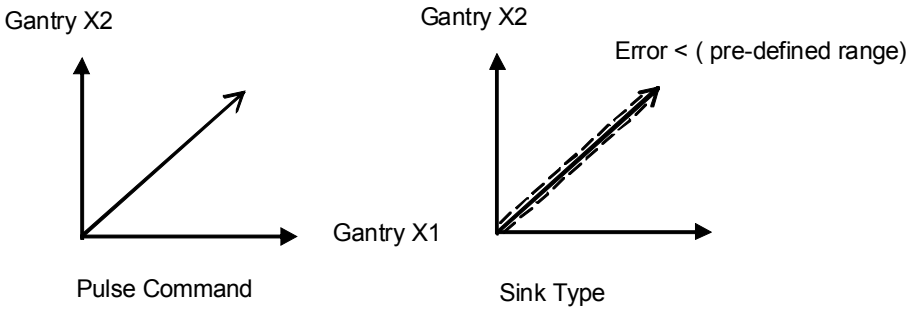
If the gantry X1 and X2 has different loading and servo parameters, these two axes will not move simultaneously.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.13.1 Closed-loop Gantry Mode

In order to solve open loop gantry problem, we use PCI-8392/ 8392H to make a closed loop system to control gantry stage. The control architechture is shown as below:





4.13.2 Error Handling of Gantry Mode

During closed loop gantry motion, there could be some error conditions to make one of the gantry axis to stop. Once this error is happened, the other axis will stop immediately. The error could be from pre-defined error tolerane overrun, emergency stop occurs, alarm from motor driver, end-limit stop and so on.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.14 HSL System Introduction (PCI-8392 Only)

4.14.1 The HSL System

The HSL is an innovative distributed I/O technology that enables time-deterministic scanning of thousands of I/O points in milliseconds using master-slave architecture. The PCI-8392H is the HSL master controller (the HSL master controller mentioned in this chapter refers to the PCI-8392H) which adopts PC-based embedded design to form compact size controller. By using commercial Ethernet cable with RJ-45 connector, you can easily set up the HSL slave modules as close as possible to the sensor devices, reducing wiring effort. Aside from the I/O modules, ADLINK provides the remote motion control module with 4-axis pulse train type. The HSL network suits a variety of machine-making applications as it integrates discrete I/O, analog I/O, thermocouple module, and motion control. This local network delivers rapid response time, time-deterministic scanning and multiple-axis control. For details on operation, please refer to ADLINK HSL User Manual for I/O operation.

The HSL system features:

- Distributed solution based on PC architecture or embedded platform
- Convenient wiring for remote distributed I/O modules, including discrete I/Os and analog I/Os
- Space-saving and discrete low-profile U-series form factor
- Hundreds of discrete I/O points
- Time-deterministic, fast scanning
- High-speed data acquisition
- Up to 60 axes of remote motion control with the PCI-8392H system
- Motion control features point table management and motion script download to enhance execution efficiency

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.14.2 Product Overview

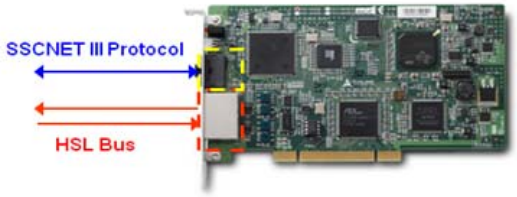


Figure 4-1: HSL Overview

4.14.3 Product Highlights

High-speed performance

With scanning speed as high as 2000 points per ms, it takes only 945 µs for an HSL master controller to scan all the discrete I/O points of slave modules under 12 Mbps. For example, a distributed control system with 63 slave I/O modules of HSL-DI16DO16-UD-NN with 2016 discrete I/O points can be scanned or updated within 945 µs.

Time-deterministic scanning

The HSL master controller implements a deterministic time period when scanning all slave I/O modules. The total scanning cycle time is exactly proportional to the number of slave indexes. At 12 Mbps, every 15.17 µs is added for another slave index. For a PCI-8392H with 30 discrete I/O slave modules (where every discrete I/O module occupies one slave index), the scanning time period is precisely 30 X 15.17 µs = 455.1 µs. The scan time unit based on transmission rate is illustrated below.

	3 Mbps	6 Mbps	12 Mbps
Full Duplex	60.67 µs	30.33 µs	15.17 µs
Half Duplex	118 µs	59 µs	29.5 µs

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Convenient wiring

The HSL master controller connects to all slave I/O modules using Ethernet cables. This dramatically reduces the wiring costs and effort. With Ethernet cables, hundreds or even thousands of I/O data can transmitted between the HSL master controller and slave I/O modules. The HSL wiring is the easiest and most cost-effective solution to date. For low profile series, you can make the connection by direct wiring.

Multiple I/O points

The PCI-8392H offers one HSL master network. If choosing all connected modules as HSL-DI16DO16-US-NN, a total of 1008 digital input and 1008 digital output points are supported.

Easy I/O expansion

Expanding I/O points for centralized configuration requires more I/O boards and available PCI or ISA slots. Problems occur when system needs more I/O points while there are no available slot. In contrast with centralized configuration, the distributed I/O configuration eliminates this limitation of a centralized I/O configuration. With the HSL system, adding more I/O points only requires one more slave I/O module and an Ethernet cable for communication link.

Self-diagnostic function

The HSL provides a self-diagnostic function that eliminates communication failures. This function continuously monitors the network status while a status register keeps the accumulated slave-no-response count for every individual slave I/O module. Also, the HSL system features the CRC12 to eliminate any communication error.

Modular design of slave I/O

ADLINK offers a variety of slave module types Including the metal-cased M series, non-metal DB series, and U series for compact systems. The M and DB series require a terminal board for connection. Terminal boards act as carrier of slave I/O module with wiring function. The Ethernet port and screw terminal on the terminal boards make it easier to replace I/O modules without turning and wiring off the system.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Remote motion control compatibility

ADLINK also offers remote motion control solution based on the HSL network Including the HSL-4XMO-CG-N/P and HSL-4XMO-CD-N/P that could connect up to four axes. The HSL-4XMO-CG-N/P features a general-type interface for use with stepper or linear motors, while the HSL-4XMO-CD-N/P has a D-sub interface. By using a transfer cable, you can connect to specific servo amplifier. You can easily make a distributed control application that includes discrete I/O, analog I/O, and remote motion control.

Easy to program

Every HSL master card comes with 32 KB SRAM that carries all the I/O status information of the HSL system. The ASIC on the HSL master board communicates with all remote slave I/O modules at fixed scanning period and keeps the most updated I/O status information on the SRAM. You can easily read/write the most updated I/O information and never worry with the HSL protocol.

4.14.4 HSL System Specifications

Software support

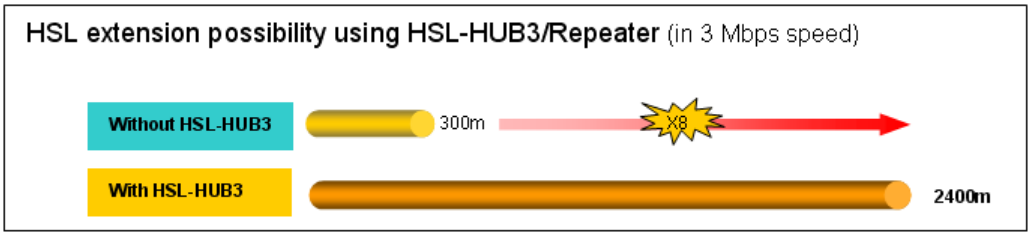
- Windows XP/2k library
- Remote operation

Transmission Speed	L (m)
3 Mbps	300
6 Mbps	200
12 Mbps	100

Table 4-10: Remote Operation

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Supports maximum 2.4 km wiring via seven HSL-HUB/ Repeater modules



	Without HUB	HUBX1	HUBX2	HUBX5	HUBX7
12 Mbps	100 m	200 m	300 m	600 m	800 m
6 Mbps	200 m	400 m	600 m	1200 m	1600 m
3 Mbps	300 m	600 m	900 m	1800 m	2400 m

Wiring

- Connector: RJ-45 (on master controller and some of slave modules)
- Cable: Cat-5 100 Base/TX Ethernet cable with shielding

Communications

- Multi-drop full-duplex RS-422 with transformer isolation scheme
- Transmission speed: 3/6/12 Mbps (6 Mbps is factory default setting).
- I/O refresh rate: scan time unit × numbers of slave indexes (minimum is 3; maximum is 63)

	3 Mbps	6 Mbps	12 Mbps
Full Duplex	60.67 µs	30.33 µs	15.17 µs
Half Duplex	118 µs	59 µs	29.5 µs

- Communication model: single master to multi-slave
- Communication method: command/response type hand-shaking
- CRC12 and dedicated protocol for eliminating communication errors

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

4.14.5 HSL Series Products

Slave I/O modules

A variety of HSL slave I/O modules are available.

Series	Model	Discrete Input	Discrete Output	Analog Input	Analog Output	Start Index Setting Range	Slave Index Occupation
DB	HSL-DI32-DB-N/P	32				(1,3, 5, …,61)	2
	HSL-DO32-DB-N/P		32			(1,3, 5, …,61)	2
	HSL-DI16DO16-DB-N/P	16	16			1-63	1
M	HSL-DI32-M-N/P	32				(1,3,…,61)	2
	HSL-DO32-M-N/P		32			(1,3,…,61)	2
	HSL-DI16DO16-M-NN/NP/PN//PP	16	16			1-63	1
	HSL-R8DI16-M-N/P	16	8 relay			1-63	1
	HSL-AI16AO2-M-VV			16	2	1-61	2
	HSL-AI16AO2-M-AV			16	2	1-61	2
U	HSL-DI16DO16-US/UJ	16	16			1-63	1
	HSL-DI16-UL	16				1-63	1
	HSL-AO4				4	1-62	2

Table 4-11: Slave I/O modules

Terminal Base

A variety of HSL terminal base are also available.

Model Numbers	Module Type Support	Module Number Support
HSL-TB64-DIN	All the HSL DB series	2
HSL-TB32-U-DIN	All the HSL DB series	1
HSL-TB32-M-DIN	All the HSL M series	1
HSL-TB32-MD	All the HSL M series	1

Table 4-12: Terminal Base

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5 MotionCreatorPro 2

After installing the hardware (Chapters 2 and 3), it is necessary to correctly configure all cards and double-check the system before running. This chapter gives guidelines for establishing a control system and manually testing the cards to verify correct operation.

The MotionCreatorPro 2 software provides a simple yet powerful means to setup, configure, test, and debug a motion control system that uses the PCI-8392, PCI-8392H, PCI-8253 and PCI-8256 cards.

Note: MotionCreatorPro 2 is only available for Windows 2000/XP operating system and a monitor with a screen resolution of 1024x768 or higher. MotionCreatorPro 2 is not available for DOS environments.

5.1 Run MotionCreatorPro 2

After installing the software drivers for the card in Windows 2000/XP, the motion creator program is located at <chosenpath>\PCI-8392\MotionCreatorPro 2. To launch the program, doubleclick on the executable file or click **Start > Program Files > ADLINK> PCI-8392 > MotionCreatorPro 2**.

5.2 About MotionCreatorPro 2

Before Running MotionCreatorPro, the following issues should be kept in mind.

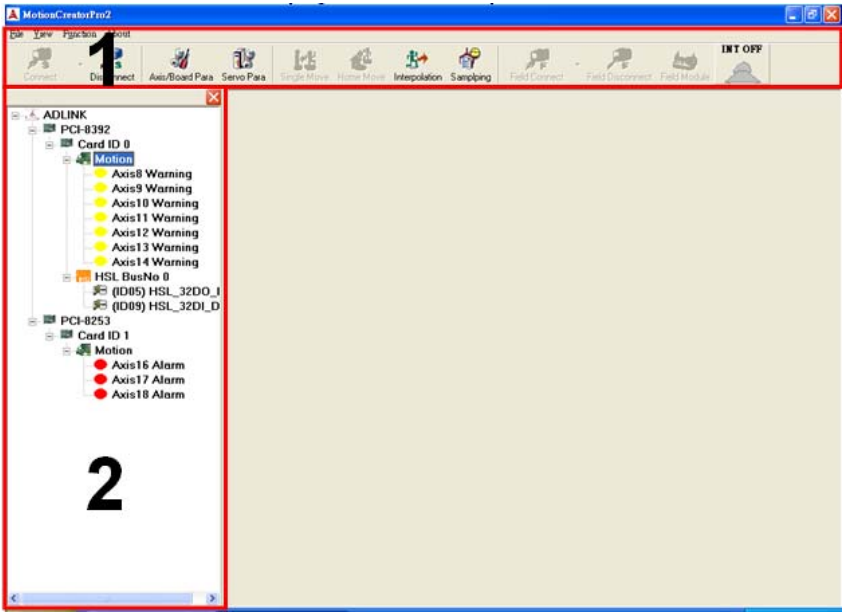
1. MotionCreatorPro is a program written in VB.NET 2003, and is available only for Windows 2000/XP with a screen resolution higher than 1024x768. It cannot be run under DOS.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.3 MotionCreatorPro 2 Introduction

5.3.1 Main Menu

The main menu appears after launching the Motion Creator Pro2. Refer to the illustration on the next page for the description of the main menu functions.



LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Functions Button: Use these buttons to select function you want test.

- Configuration

Button	Function	Description
	SSCNET Connect	Connect axis. Please select baud rate at button right side and connect. ***It's only for PCI-8392***
	SSCNET Disconnect	Disconnect axis. ***It's only for PCI-8392***
	Axis/Board Configuration	Configure axis or board parameter.
	SSCNET Servo Parameter Configuration	Configure parameter of MITSUBINSHI J3 Servo Driver. ***It's only for PCI-8392***

▶ Movement

Button	Function	Description
	Single Movement	Single Axis movement (PTP), include absolute and relative function.
	Home Return Movement	Home return movement.
	Interpolation	Interpolation function.
	Sampling	Sampling function, it can select source and drew its profile.

▶ HSL

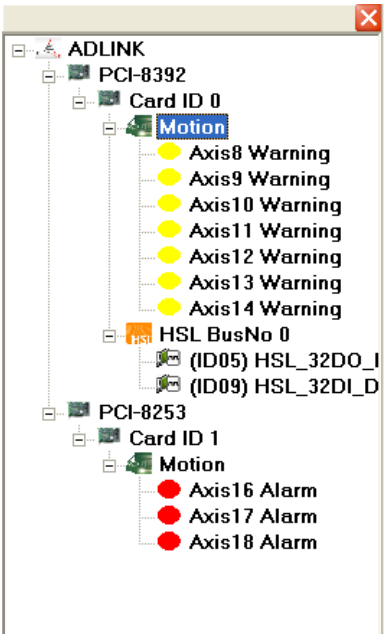
Button	Function	Description
	HSL Connect	Connect HSL module. Please select baud rate at button right side and connect. ***It's only for PCI-8392***
	HSL Disconnect	Disconnect HSL module. ***It's only for PCI-8392***
	HSL Module Test	If connect correctly, select module and use this to do module test. ***It's only for PCI-8392***

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

▶ Others

Button	Function	Description
	Motion Interrupt On/Off	Use this button to control motion interrupt enable/disable.

Product Tree View: This area show how many PCI-8392, PCI-8253/56 cards and axes.

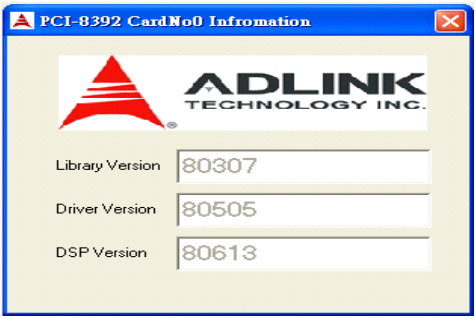


ICON	Function	Description
(Yellow)	Warning	Servo Warning
(Red)	Alarm	Servo Alarm
(Black)	Normal (Servo OFF)	No Error and servo off
(Green)	Normal (Servo ON)	No Error and servo on

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

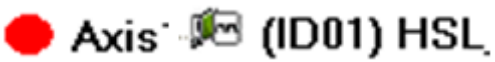
5.3.2 Card Number Layer

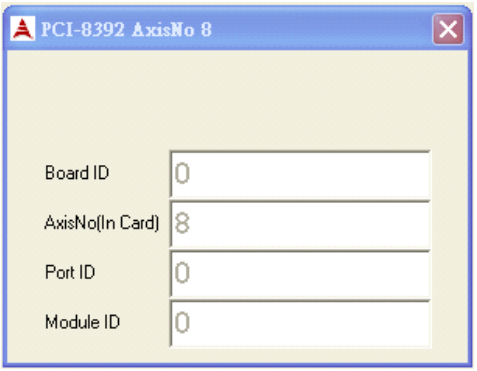
Card ID

Double Click Left Button	Description
	**1.Library Version:** Dll version **2.Driver Version:** WDM driver Version **3.DSP Version:** DSP firmware version
Right Botton	**Description**
	**1. Servo On All:** Make all motors in card to servo On. **2. Servo Off All:** Make all motors in card to servo off.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.3.3 Axis/Module Layer

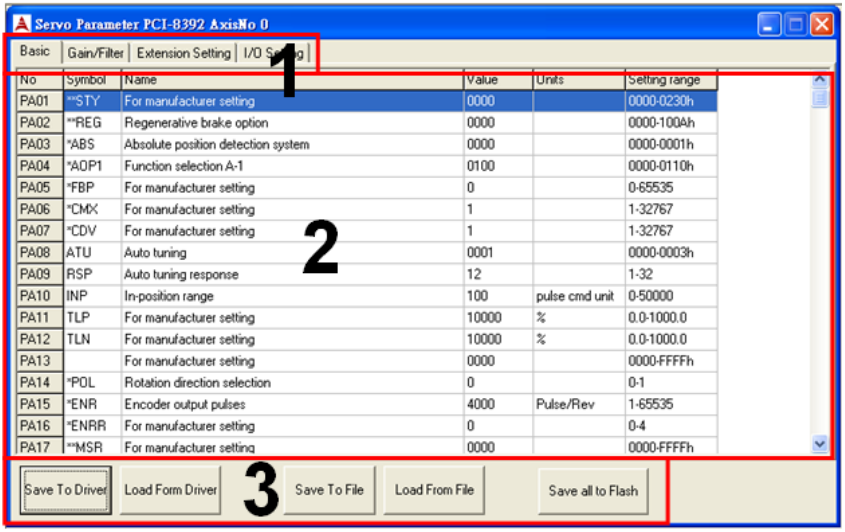


Double Click Left Button	Description
	**1. Board ID:** Card Number **2. Axis No(In Card):** Axis number in card (not system) **3.Port ID:**HSL Port ID **4.Module ID:**HSL module ID

Right Botton	Description
	**1. Servo On:** Make motor servo On. **2. Servo Off:** Make motor servo Off. **3. Alarm Reset:** Use this to reset alarm signal. ***Only for PCI-8392.***

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.3.4 Servo Parameter Configuration

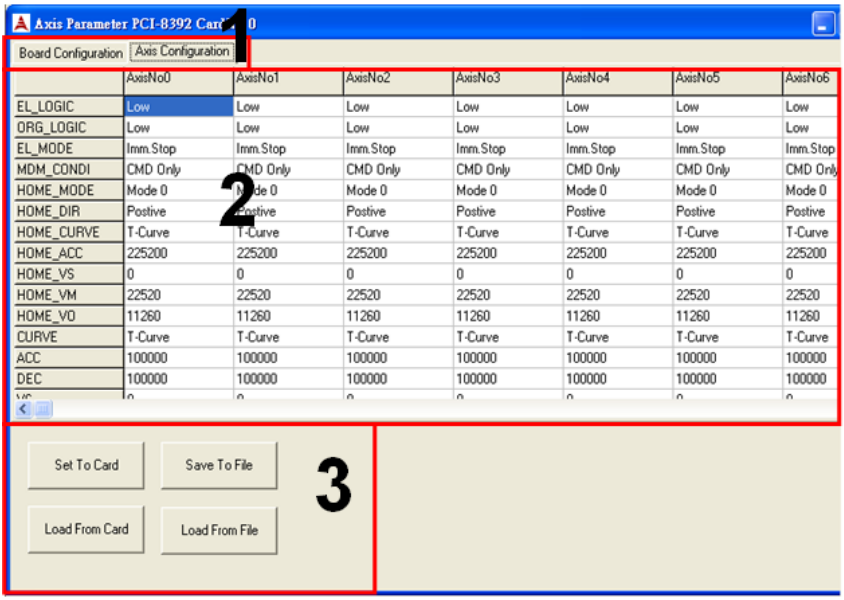


***This function is only for PCI-8392.**

1. **Parameter Type Select**: There are four types can be selected: Basic, Gain/Filter, Extension and IO Setting.
2. **Parameter Table**: View and modify parameter.
3. **Buttons**: There are five buttons available.
   - ▷ **Save To Driver**: Save Parameter to servo driver.
   - ▷ **Load Form Driver**: Load Parameter form servo driver.
   - ▷ **Save To File**: Save setting to file.
   - ▷ **Load Form File**: Load setting to file.
   - ▷ **Save all to Flash**: Save setting to Flash ROM on PCI-8392.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.3.5 Axis/Board Configuration

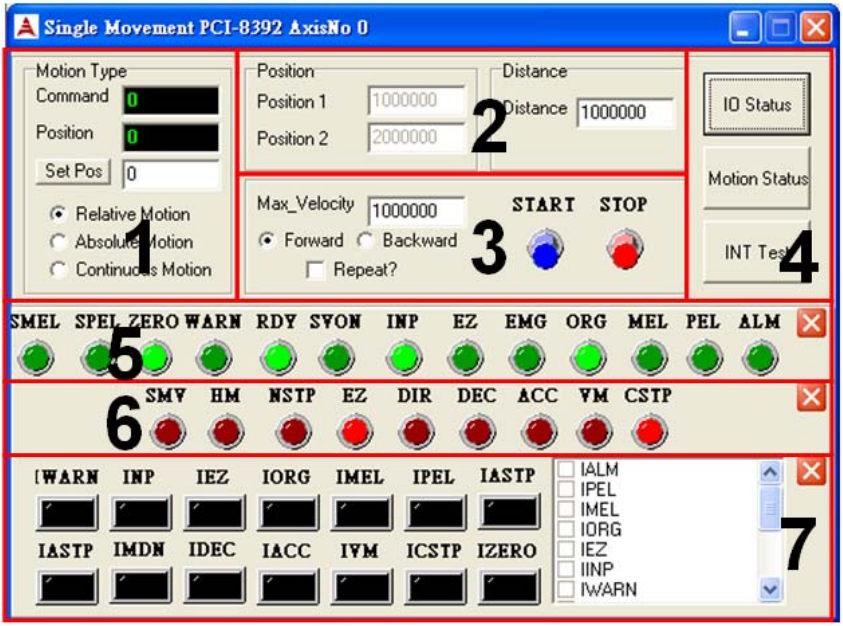


	AxisNo0	AxisNo1	AxisNo2	AxisNo3	AxisNo4	AxisNo5	AxisNo6
EL_LOGIC	Low	Low	Low	Low	Low	Low	Low
ORG_LOGIC	Low	Low	Low	Low	Low	Low	Low
EL_MODE	Imm.Stop	Imm.Stop	Imm.Stop	Imm.Stop	Imm.Stop	Imm.Stop	Imm.Stop
MDM_CONDI	CMD Only	CMD Only	CMD Only	CMD Only	CMD Only	CMD Only	CMD Only
HOME_MODE	Mode 0	Mode 0	Mode 0	Mode 0	Mode 0	Mode 0	Mode 0
HOME_DIR	Postive	Postive	Postive	Postive	Postive	Postive	Postive
HOME_CURVE	T-Curve	T-Curve	T-Curve	T-Curve	T-Curve	T-Curve	T-Curve
HOME_ACC	225200	225200	225200	225200	225200	225200	225200
HOME_VS	0	0	0	0	0	0	0
HOME_VM	22520	22520	22520	22520	22520	22520	22520
HOME_VO	11260	11260	11260	11260	11260	11260	11260
CURVE	T-Curve	T-Curve	T-Curve	T-Curve	T-Curve	T-Curve	T-Curve
ACC	100000	100000	100000	100000	100000	100000	100000
DEC	100000	100000	100000	100000	100000	100000	100000

1. **Parameter Type Select**: There are two types available: Board and Axis.
2. **Parameter Table**: View and modify parameter.
3. **Buttons**: There are four buttons available.
   - ▷ **Set to Card**: Save Parameter to Flash ROM on card .
   - ▷ **Load Form Card**: Load Parameter form Flash ROM on card
   - ▷ **Save To File**: Save settings to file.
   - ▷ **Load Form File**: Load settings to file.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.3.6 Single Movement



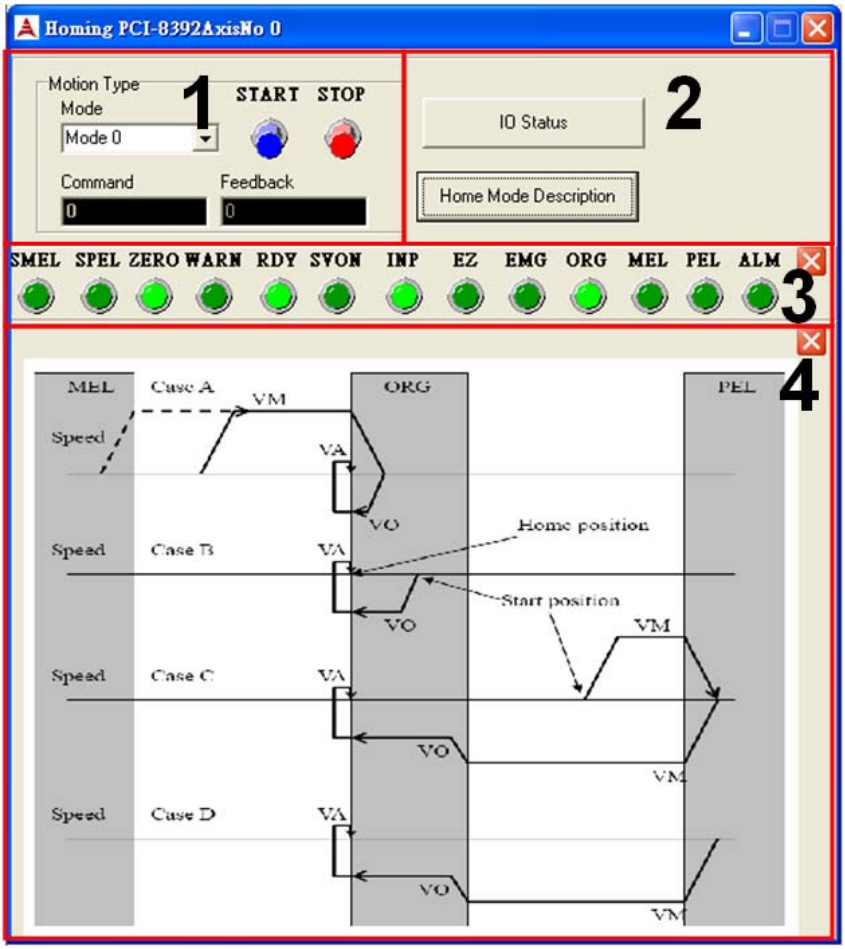
1. **Position**:
   - ▷ **Command**: Displays the value of the command counter.
   - ▷ **Feedback**: Displays the value of the feedback position counter.
   - ▷ **Position** Reset: Clicking this button will set all positioning counters to a specified value.
   - ▷ **Absolute Motion**: “Position1” and “position2” will be used as absolution target positions for motion.
   - ▷ **Relative Motion**: “Distance” will be used as relative displacement for motion.
   - ▷ **Continuous Motion**: Velocity motion mode.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

2. **Motion Parameters**: Set the parameters for single axis motion.
   - ▷ **Position**: Set the absolute position for “Absolute Mode.” It is only effective when “Absolute Mode” is selected.
   - ▷ **Distance**: Set the relative distance for “Relative Mode.” It is only effective when “Relative Mode” is selected.
3. **Play Key**:
   - ▷ **Start**: Clicking this button will cause the card start to move.
   - ▷ **Stop**: Clicking this button will cause the card stop move.
   - ▷ **Maximum Velocity**: Set the maximum velocity of motion in units of PPS. In “Absolute Mode” or “Relative Mode,” only the value is effective.
   - ▷ **Repeat Mode**: When “On” is selected, the motion will become repeat mode (forward<->backward or position1<->position2). It is only effective when “Relative Mode” or “Absolute Mode” is selected.
   - ▷ **Forward/Backward**: Direction Selects.
4. **Buttons**:
   - ▷ **IO Status**: Expand IO Status indicator.
   - ▷ **Motion Status**: Expand IO Status indicator.
   - ▷ **INT Test**: Expand INT test form.
5. **IO Status**: Show mechanical IO status.
6. **Motion Status**: Show Motion Status.
7. **INT Setting and status**: Use check list box right side to configure INT factor, and check INT status.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

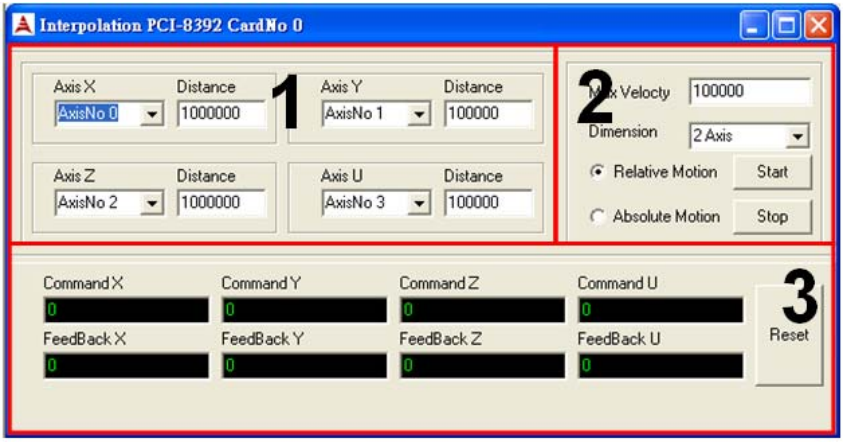
5.3.7 Home Return



1. **Position and Play Key**:
   - ▷ **Command**: Show the value of the command counter.
   - ▷ **Feedback**: Show the value of the feedback position counter.
   - ▷ **Start button**: Clicking this button will cause the card start home move.
   - ▷ **Stop button**: Clicking this button will cause the card stop home move.
   - ▷ **Mode selection**: Select home mode.
   - ▷ **Maximum Velocity**: Set the maximum velocity of motion in units of PPS. In “Absolute Mode” or “Relative Mode,” only the value is effective.
2. **Buttons**:
   - ▷ **IO Status**: Expand IO Status indicator.
   - ▷ **Home mode description**: Expand home mode description dialog..
3. **IO Status**: Show mechanical IO status.
4. **Home mode description**: Show home mode description.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

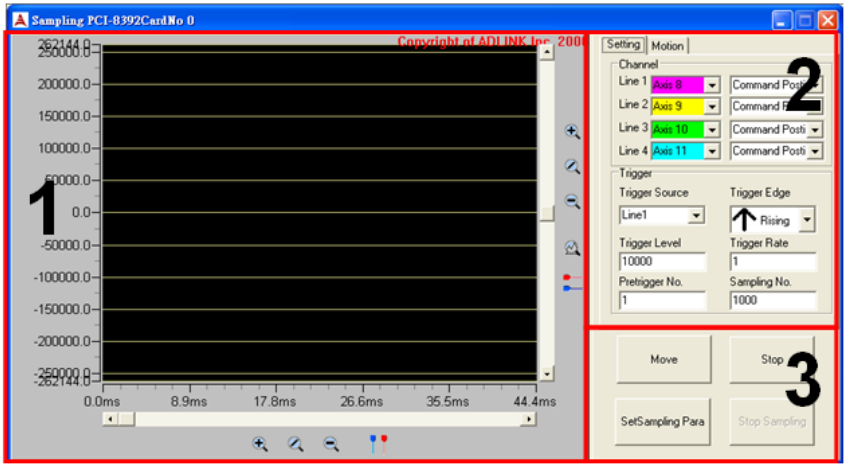
5.3.8 Interpolation



1. **Axis Select and Distance Setting**: Select axis and set distance for each axis to do interpolation.
2. **Movement configuration**:
   - ▷ **Max Velocity**: Set maximum velocity for movement.
   - ▷ **Dimension**: Select dimension (axis) to do interpolation, minimum dimension is two.
   - ▷ **Absolute Motion**: “Position1” and “position2” will be used as absolution target positions for motion.
   - ▷ **Relative Motion**: “Distance” will be used as relative displacement for motion.
   - ▷ **Start**: Clicking this button will cause the card start to move.
   - ▷ **Stop**: Clicking this button will cause the card stop move.
3. **Position**
   - ▷ **Command**: Show the value of the command counter.
   - ▷ **Feedback**: Show the value of the feedback position counter.
   - ▷ **Reset**: Reset all command and feedback counters.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.3.9 Sampling

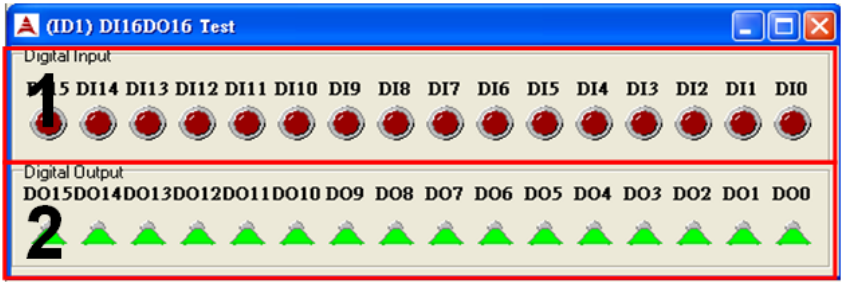


1. **Profile Area**: Turning data will show as profile here.
2. **Sampling and Motion Setting**: Select setting for sampling configuration and select motion to do movement configuration.
3. **Sampling Setting**:
   - ▷ **Channel**: Select one of the channel signals to be the trigger source
   - ▷ **Trigger Level**: Trigger value
   - ▷ **Trigger Edge**: Specify the rising edge or falling edge trigger
   - ▷ **Sampling No**: Total amount of the gathering data
   - ▷ **Pretrigger No**: Amount of the pretrigger data
4. **Buttons**:
   - ▷ **Move**: Make axis to move.
   - ▷ **Stop**: Make axis to stop.
   - ▷ **SetSampling Para**: Start Sampling.
   - ▷ **Stop Sampling**: Stop Sampling.

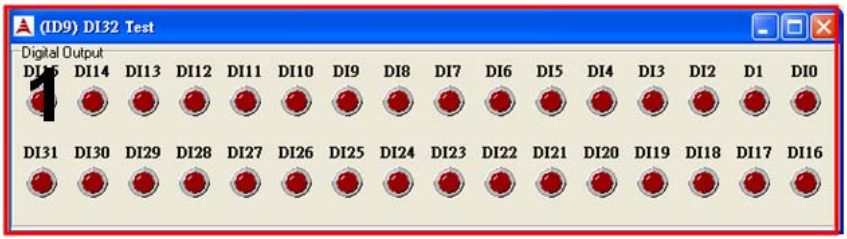
LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

5.4 HSL Module Function Test

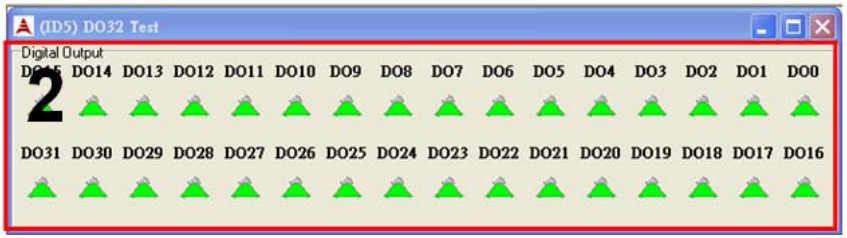
5.4.1 Module Type1: HSL-DI16DO16-N/P



5.4.2 Module Type2: HSL-DI32-N/P



5.4.3 Module Type2: HSL-DO32-N/P



1. DI Function Indicator: It will turn on if DI input.
2. DO Function Controller: Switch DO function turn on or turn off.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6 Function Library

6.1 C/C++ Programming Library

This section details all the functions. The function prototypes and some common data types are decelerated in **ASP168.h**. We suggest you to use these data types in your application programs. The following table shows the data type names and their range.

Type Name	Description:	Range
U8	8-bit ASCII character	0 to 255
I16	16-bit signed integer	-32768 to 32767
U16	16-bit unsigned integer	0 to 65535
I32	32-bit signed long integer	-2147483648 to 2147483647
U32	32-bit unsigned long integer	0 to 4294967295
F32	32-bit single-precision floating-point	-3.402823E38 to 3.402823E38
F64	64-bit double-precision floating-point	-1.797683134862315E308 to 1.797683134862315E309
Boolean	Boolean logic value	TRUE, FALSE

The functions of the PCI-8392's software drivers use full-names to represent the functions real meaning. The naming convention rules are:

In a 'C' programming environment:

APS_{action_name}. e.g. **`APS_initial`**().

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.2 List of Functions

This chapter describes the supporting software for the PCI-8392/8392H card. User can use these functions to develop programs in C, C++, or Visual Basic. If Delphi is used as the programming environment, it is necessary to transform the header files, APS168.h manually.

Sec	Function name	Description	Page
6.3	**System & Initialization**		127
	APS_initial	Device initialization	127
	APS_close	Device close	129
	APS_version	Get the version of the library	130
	APS_device_driver_version	Get the driver's version of devices	131
	APS_get_axis_info	Get the information of the specified axis	132
	APS_set_board_param	Set board parameter	134
	APS_get_board_param	Get board parameter	135
	APS_set_axis_param	Set axis parameter	136
	APS_get_axis_param	Get axis parameter	137
	APS_get_watch_timer	Get system watch timer counter	138
6.4	**SSCNET function**		139
	APS_start_sscnet	Start the network of SSCNET	139
	APS_stop_sscnet	Stop the network of SSCNET	140
	APS_get_sscnet_servo_param	Read current servo parameter value	141
	APS_set_sscnet_servo_param	Set servo parameter	143
	APS_get_sscnet_servo_alarm	Get current servo alarm information	145
	APS_reset_sscnet_servo_alarm	Servo alarm reset	147
	APS_save_sscnet_servo_param	Save servo parameter to flash ROM	148
	APS_get_sscnet_servo_abs_position	Get absolute reference position from servo driver	150
	APS_save_sscnet_servo_abs_position	Save absolute reference position to flash ROM	152
	APS_load_sscnet_servo_abs_position	Load absolute reference position from flash ROM	154

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Sec	Function name	Description	Page
6.5	Motion IO and motion status		156
	APS_motion_status	Return motion status	156
	APS_motion_io_status	Return motion IO status	158
	APS_set_servo_on	Set servo ON/OFF	159
	APS_get_position	Get feedback position	160
	APS_set_position	Set feedback position	161
	APS_get_command	Get command position	162
	APS_set_command	Set command position	163
6.6	Single axis motion		164
	APS_relative_move	Begin a relative distance move	164
	APS_absolute_move	Begin a absolute position move	166
	APS_velocity_move	Begin a velocity move	168
	APS_home_move	Begin a home move	170
	APS_set_jog_param	Set Jog parameters	172
	APS_get_jog_param	Get Jog parameters	174
	APS_jog_mode_switch	Enable / Disable jog move	176
	APS_jog_start	Start / stop jog move	177
	APS_stop_move	Stop move	178
	APS_emg_stop	Emergency stop	179
6.7	Interpolation		180
	APS_absolute_linear_move	Begin an absolute position linear interpolation	180
	APS_relative_linear_move	Begin a relative distance linear interpolation	182
	APS_absolute_arc_move	Begin an absolute position circular interpolation	184
	APS_relative_arc_move	Begin a relative distance circular interpolation	187
6.8	Interrupt		190
	APS_int_enable	Interrupt main switch	190
	APS_set_int_factor	Enable/Disable interrupt factor and get interrupt handle.	192
	APS_get_int_factor	Get interrupt factor enable or disable	194
	APS_wait_single_int	Wait single interrupt event	196
	APS_wait_multiple_int	Wait multiple interrupt events	198
	APS_reset_int	Reset interrupt event to non-signaled state.	201
	APS_set_int	Set interrupt event to signaled state.	203

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Sec	Function name	Description	Page
6.9	Sampling		205
	APS_set_sampling_param	Set sampling parameter.	205
	APS_get_sampling_param	Get sampling parameter.	207
	APS_wait_trigger_sampling	Waiting for sample data.	209
6.10	Point table motion		211
	APS_set_point_table	Set point table move parameters	211
	APS_get_point_table	Get point table move parameters	214
	APS_point_table_move	Start a point table move	217
	APS_get_running_point_index	Get running point move index when axis is perform a point move	220
	APS_get_start_point_index	Get the first point move index when axis is perform a point move	222
	APS_get_end_point_index	Get the last point move index when axis is perform a point move	223
	APS_set_table_move_pause	Pause point table move	225
	APS_set_table_move_repeat	Set point table move repeat	227
6.11	Field bus functions		229
	APS_set_field_bus_param	Set field bus related parameters	229
	APS_get_field_bus_param	Get field bus related parameters	230
	APS_start_field_bus	Start the network of specified field bus	231
	APS_stop_field_bus	Stop the network of specified field bus	233
	APS_field_bus_d_set_output	Set field bus digital output	234
	APS_field_bus_d_get_output	Get field bus digital output	236
	APS_field_bus_d_get_input	Get field bus digital input	238
6.12	Definition tables		240
	Board parameters definition table		241
	Axis parameters definition table		242
	SSCNET Axis parameters definition table		244
	Sampling parameters definition table		245
	Sampling source definition table		246
	The bit definition of motion IO status		247
	Motion status definition table:		248
	Interrupt factor table:		250
	Function Return Code		252

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.3 Initialzation

6.3.1 APS_initial

Device initialization

Description:

This function is used to initialize all products on local controller supported by APS function library. It allocates system hardware resources for each board including I/O address, memory address, IRQ and DMA if needed. It retrieves a board ID for each board which is assigned by on-board switch or operating system. The board ID is a unique number for the board in the system. It is used for any other APS functions to access corresponding hardware.

If users choose on-board switch (Mode = manual ID) initial mode and there are some boards don't support this feature in the system, the board ID of these boards will be arranged after the boards having on-board switch automatically. The card ID (dip-switch) cannot be set the same when you used "manual-ID" or the function will return error.

Syntax:

```
I32 APS_initial(I32 *BoardID_InBits, I32 Mode);
APS_initial (BoardID_InBits As Long, ByVal Mode
     As Long) As Long
```

Parameters:

`I32 * BoardID_InBits`: Card ID information in bit format.

Example: If the value of BoardID_InBits is 0x11 which means that there are 2 cards in your system and those card's ID are 0 and 4.

`I32 Mode`: Enable the On board dip switch (**SW1**) to decide the Card ID. [0:By system assigned, 1:By dip switch]

Return Values:

I32 error codes: Please refer to the error code table on page 253.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
I32 ret; // return value
I32 BoardID_InBits;
I32 Mode = 0; //By system assigned
ret = APS_initial( &BoardID_InBits, Mode);

…// Do something
ret = APS_close(); //Close all cards in the
     system
```

See also:

```
APS_close();
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.3.2 APS_close

Close devices.

Description:

This function is used to close all resources allocated by APS library. The resources include system hardware resource like I/O address, memory address, IRQ and DMA. It also deletes some objects, handles or memory allocated by APS library.

Syntax:

```
I32 APS_close()
APS_close() As Long
```

Parameters:

No parameters.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

```
I32 ret; // return value
I32 BoardID_InBits;
I32 Mode = 0; //By system assigned
ret = APS_initial( &BoardID_InBits, Mode);

…// Do something
ret = APS_close(); //Close all cards in the
      system
```

See also:

```
APS_initial();
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.3.3 APS_version

Obtain the version of the library

Description:

This function is used to obtain the APS library (DLL) version information.

Syntax:

```
I32 APS_version();
APS_version() As Long
```

Parameters:

Return Values:

Return library (DLL) version.

Example:

See also:

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.3.4 APS_device_driver_version

Obtain the driver's version of devices

Description:

This function is used to get device driver version information of one board. The version information is the same for one type of board in system.

Syntax:

```
I32 APS_device_driver_version( I32 Board_ID )
APS_device_driver_version( ByVal Board_ID As Long
     ) As Long
```

Parameters:

`I32 Board_ID`: ID of the target controller. It's retrieved by successful call to **`APS_initial`**().

Return Values:

Positive value: The device driver version number,

Negative value: Error code: Please refer to error code table.

Example:

See also:

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.3.5 APS_get_axis_info

Obtain information of a specified axis

Description:

This function is used to get information of one axis. The information includes attached board ID, serial port ID, serial module ID and module type. There are two categories for axis ID index: parallel and serial types.

The parallel type axis ID indexing rule is according to the board ID. The formula is:

```
Axis ID = Board ID x Maximum number of axis
     within one board + Axis No
```

The **Axis No** parameter is the axis number within the board. The **Maximum number of axis within one board** parameter is an inside system variable of APS library. The default value is 16. If the system has some boards without axes, it still counts the formula when indexing. It is useful for users to remove/add some boards from system without rearranging axis index.

The serial type axis ID indexing rule is according to the module ID and assigned with a starting axis ID first. The formula of serial port axis would be:

```
Axis ID = Module ID x Maximum number of axis
     within one module + Starting Axis ID of port
     + Axis No
```

The **Axis No** parameter is the axis number within the module. The **Maximum number of axis within this module** parameter is an inside port variable of APS library. The default value is 4. **Starting Axis ID of port** parameter is the starting axis ID of one port assigned by users. The default value is 0. If the port has some modules without axis, it still counts the formula when indexing. This is the same with parallel type. It is useful for users to remove/add some boards from system without rearranging axis index.

SSCNET board is regarded as parallel type in axis indexing.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Syntax:

```
I32 APS_get_axis_info( I32 Axis_ID, I32
     *Board_ID, I32 *Port_ID, I32 *Module_ID );
APS_get_axis_info(ByVal Axis_ID As Long, Board_ID
     As Long, Port_ID As Long, Module_ID As Long)
     As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535

I32 *Board_ID: The returned board ID for the Axis ID. Range is from 0 to 31.

I32 *Port_ID: The returned field bus port ID of board for the axis. Range is from 1 to 15.

I32 *Module_ID: The returned module ID of port for the axis. The range is from 1-65535. *Module_ID=0 means no serial port exists.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

See also:

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.3.6 APS_set_board_param

Set board parameter

Description:

This function is used for set all kinds of parameter which has relationship with a board. Please refer to the board parameter table for the definition and detail descriptions.

Syntax:

```
I32 APS_set_board_param( I32 Board_ID, I32
     BOD_Param_No, I32 BOD_Param );
APS_set_board_param (ByVal Board_ID As Long,
     ByVal BOD_Param_No As Long, ByVal BOD_Param
     As Long) As Long
```

Parameters:

`I32 Board_ID`: The Board's ID from 0 to 31.

`I32 BOD_Param_No`: Board parameter number. please refer the board parameter table for definition.

`I32 BOD_Param`: Board parameter value. Refer to the board parameter table for detail.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

See also:

```
APS_get_board_param()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.3.7 APS_get_board_param

Obtain board parameters

Description:

This function is used for get all kinds of parameter which has relationship with a board. Please refer to the board parameter table for the definition and detail descriptions.

Syntax:

```
I32 APS_get_board_param( I32 Board_ID, I32
     BOD_Param_No, I32 *BOD_Param );
APS_get_board_param (ByVal Board_ID As Long,
     ByVal BOD_Param_No As Long, BOD_Param As
     Long) As Long
```

Parameters:

`I32 Board_ID`: The Board's ID from 0 to 31.

`I32 BOD_Param_No`: Board parameter number. please refer the board parameter table for definition.

`I32 *BOD_Param`: The returned board parameter value. Refer to board parameter table.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

See also:

```
APS_set_board_param()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.3.8 APS_set_axis_param

Set axis parameters

Description:

This function is used for set all kinds of parameter of one axis. The parameters include run mode, acceleration rate, deceleration rate, Jerk, motion I/O logic and so on. Please refer to the axis parameter table for the definition and detail descriptions.

Syntax:

```
I32 APS_set_axis_param( I32 Axis_ID, I32
     AXS_Param_No, I32 AXS_Param );
APS_set_axis_param(ByVal Axis_ID As Long, ByVal
     AXS_Param_No As Long, ByVal AXS_Param As
     Long) As Long
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535.

`I32 AXS_Param_No`: Axis parameter number from 0 to 65535. Each parameter is defined by a unique symbol in 3-6 characters. Refer to axis parameter table.

`I32 AXS_Param`: Axis parameter value. Refer to axis parameter table

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

See also:

```
APS_get_axis_param()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.3.9 APS_get_axis_param

Obtain axis parameters

Description:

This function is used for set all kinds of parameter of one axis. The parameters include run mode, acceleration rate, deceleration rate, jerk, motion I/O logic and so on. Please refer to the axis parameter table for the definition and detail descriptions.

Syntax:

```
I32 APS_get_axis_param( I32 Axis_ID, I32
     AXS_Param_No, I32 *AXS_Param );
APS_get_axis_param (ByVal Axis_ID As Long, ByVal
     AXS_Param_No As Long, AXS_Param As Long) As
     Long
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535.

`I32 AXS_Param_No`: Axis parameter number from 0 to 65535. Each parameter is defined by a unique symbol in 3-6 characters. Refer to axis parameter table.

`I32 *AXS_Param`: Axis parameter value. Refer to axis parameter table

Return Values:

I32 Error code: Please refer to error code table.

Example:

See also:

```
APS_set_axis_param();
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.3.10 APS_get_watch_timer

Obtain the system watch timer counter

Description:

This function is used to get system counter. When system is power on, the system counter will count up every system cycle. Users could use this function to watch system is under control.

Syntax:

```
I32 APS_get_watch_timer( I32 Board_ID, I32 *Timer
     );
APS_get_watch_timer( ByVal Board_ID As Long,
     Timer As Long ) As Long
```

Parameters:

`I32 Board_ID`: The Board's ID from 0 to 31.

`I32 *Timer`: return system timer.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

See also:

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.4 SSCNET function

6.4.1 APS_start_sscnet

Start the SSCNET network

Description:

This function is used to start SSCNET networking. Once it is started, the SSCNET will start to search the servo drivers connected to the network. It returns axis connecting status inside the bit of the 32-bit value. This function will hold until SSCNET communication established when users issue the function.

Some SSCNET parameters should be set before start the network such as SSCNET cycle time and so on. Please refer to the SSCNET parameter table for the detail description.

Syntax:

```
I32 APS_start_sscnet( I32 Board_ID, I32
     *AxisFound_InBits );
APS_start_sscnet (ByVal Board_ID As Long,
     AxisFound_InBits As Long) As Long
```

Parameters:

`I32 Board_ID`: ID of the target controller. It's retrieved by successful call to APS_initial().

`I32 *AxisFound_InBits`: The returned connected axis in bit.

Eg. AxisFound_InBits = 0x111 means Axis switch index: 0, 4 and 8 are connected on line.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

See also:

```
APS_stop_sscnet ()
APS_set_board_param()
APS_get_board_param()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.4.2 APS_stop_sscnet

Stop the SSCNET network

Description:

This function is used to stop SSCNET networking. Once it is stopped, the SSCNET will stop communicating the servo drivers and all servo drivers will be free running after that.

Syntax:

```
I32 APS_stop_sscnet( I32 Board_ID );
APS_stop_sscnet (ByVal Board_ID As Long) As Long
```

Parameters:

`I32 Board_ID`: ID of the target controller. It's retrieved by successful call to APS_initial().

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

See also:

```
APS_start_sscnet()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.4.3 APS_get_sscnet_servo_param

Read current servo parameter value

Description:

This function is used to get servo parameters from servo driver. User can read two servo parameters at once. It also can read only one parameter using Para_No1. If you set **`Para_No2 = 0`**, **`Para_dat2`** can be set to null.

This function is valid only after SSCNET network is started.

Never try to change parameters for manufacturer setting.

The definition of servo parameter please refer to Mitsubishi J3-B manual.

Syntax:

```
I32 APS_get_sscnet_servo_param( I32 Axis_ID, I32
     Para_No1, I32 *Para_Dat1, I32 Para_No2, I32
     *Para_Dat2 );
APS_get_sscnet_servo_param(ByVal Axis_ID As Long,
     ByVal Para_No1 As Long, Para_Dat1 As Long,
     ByVal Para_No2 As Long, Para_Dat2 As Long)
     As Long
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535.

`I32 Para_No1`: Servo parameter Number. The parameter meaning, please refer to the manual of servo driver.

Format: 0 x 0 N XX

N: PA: 0, PB: 1, PC: 2, PD: 3

XX: parameter number.

Eg. 0x0107: PB07, 0x000A: PA10, 0x020F

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

I32 *Para_Dat1:

I32 Para_No2: Servo parameter Number. The parameter meaning, please refer to the manual of servo driver.

Format: 0 x 0 N XX

N: PA: 0, PB: 1, PC: 2, PD: 3

XX: parameter number.

Eg. 0x0107: PB07, 0x000A: PA10, 0x020F

I32 *Para_Dat2: Pointer of I32 variable. When **Para_No2** is set to 0, The **Para_Dat2** could be set to null (0).

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

See also:

```
APS_set_sscnet_servo_param();
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.4.4 APS_set_sscnet_servo_param

Set servo parameters

Description:

This function is used to set servo parameters to servo driver. User can read two servo parameters at once. It also can read only one parameter using Para_No1. If users set Para_No2 = 0, Para_dat2 can be set to null.

This function is valid only after SSCNET network is started.

Some servo parameters change is invalid after network is start. User should restart the network. The parameter change will be active.

For the definition of servo parameters please refer to the Mitsubishi J3-B manual.

Syntax:

```
I32 APS_set_sscnet_servo_param( I32 Axis_ID, I32
     Para_No1, I32 Para_Dat1, I32 Para_No2, I32
     Para_Dat2 );
APS_set_sscnet_servo_param(ByVal Axis_ID As Long,
     ByVal Para_No1 As Long, ByVal Para_Dat1 As
     Long, ByVal Para_No2 As Long, ByVal
     Para_Dat2 As Long) As Long
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 Para_No1: Servo parameter Number. The parameter meaning, please refer to the manual of servo driver.

Format: 0 x 0 N XX

N: PA: 0, PB: 1, PC: 2, PD: 3

XX: parameter number.

Eg. 0x0107: PB07, 0x000A: PA10, 0x020F

I32 Para_Dat1:

I32 Para_No2: Servo parameter Number. The parameter meaning, please refer to the manual of servo driver.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Format: 0 x 0 N XX

N: PA: 0, PB: 1, PC: 2, PD: 3

XX: parameter number.

Eg. 0x0107: PB07, 0x000A: PA10, 0x020F

`I32 Para_Dat2`: Servo parameter data. When Para_No2 is set to 0, The Para_Dat2 could be set to null (0).

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

See also:

```
APS_get_sscnet_servo_param();
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.4.5 APS_get_sscnet_servo_alarm

Get current servo alarm information

Description:

This function is used to get alarm number when servo alarm occurs. The alarm information include alarm number and alarm detail. Please refer to servo driver manual for the detail description.

When a servo alarm occurred, you should use this function before reset alarm otherwise the alarm information will be reset.

Syntax:

```
I32 APS_get_sscnet_servo_alarm( I32 Axis_ID, I32
     *Alarm_No, I32 *Alarm_Detail );
APS_get_sscnet_servo_alarm(ByVal Axis_ID As Long,
     Alarm_No As Long, Alarm_Detail As Long) As
     Long
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535.

`I32 *Alarm_No`: Alarm number. Please refer to servo driver manual.

`I32 *Alarm_Detail`: Alarm detail. Please refer to servo driver manual.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
I32 Alarm_No;
I32 Alarm_Detail;

…//Alarm occurred!
APS_get_sscnet_servo_alarm(Axis_ID, &Alarm_No,
     &Alarm_Detail ); //Get alarm informaiton
…//Remove the alarm cause
APS_reset_sscnet_servo_alarm(Axis_ID ); //Reset
     servo alarm
...
```

See also:

```
APS_reset_sscnet_servo_alarm();
```

6.4.6 APS_reset_sscnet_servo_alarm

Servo alarm reset

Description:

When servo alarm occurs, servo motor will stop moving. After the alarm condition passed, this function can help to clear alarm and reset servo.

Syntax:

```
I32 APS_reset_sscnet_servo_alarm( I32 Axis_ID );
APS_reset_sscnet_servo_alarm(ByVal Axis_ID As
     Long) As Long
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

```
I32 Alarm_No;
I32 Alarm_Detail;

…//Alarm occurred!
APS_get_sscnet_servo_alarm(Axis_ID, &Alarm_No,
     &Alarm_Detail ); //Get alarm informaiton
…//Remove the alarm cause
APS_reset_sscnet_servo_alarm(Axis_ID ); //Reset
     servo alarm...
```

See also:

```
APS_get_sscnet_servo_alarm();
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.4.7 APS_save_sscnet_servo_param

Save servo parameter to flash

Description:

This function is used to save servo parameter from SRAM to flash memory on the controller card.

When the system (controller) is power on, it will copy default servo parameter[1] to SRAM. After SSCNET network is established, controller will get the servo parameter from connected servo drives. In other word, the servo parameter will be synchronized between servo drivers and controller after start SSCNET network. The parameter is remained default, if the Axis is null (The axis ID doesn't be used).

Servo parameters of all axes (16 axes) will be saved at once when you issue this function. You cannot save every servo driver' parameter separately.

Syntax:

```
I32 APS_save_sscnet_servo_param( I32 Board_ID );
APS_save_sscnet_servo_param(ByVal Board_ID as
     Long) As Long
```

Parameters:

`I32 Board_ID`: ID of the target controller. It's retrieved by successful call to APS_initial().

Return Values:

I32 error codes: Please refer to the error code table on page 253.

1. The default servo parameter is defined in servo driver manual.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
// Config servo parameter
// APS_set_sscnet_servo_param …
APS_save_sscnet_servo_param( Board_ID ); //Save
     servo parameter to flash.
```

See also:

APS_set_sscnet_servo_param
APS_get_sscnet_servo_param,

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.4.8 APS_get_sscnet_servo_abs_position

Get absolute reference position from servo driver

Description:

This function is used to get the absolute position (encoder) of SSCNET servo driver. This function can be issued only when SSCNET network is started. This function is for absolute position coordinate system. After user perform a home return operation, the user must issue this function to get absolute position data. In the meantime, and kernel will copy the absolute position of servo drive to axis parameter. Therefore, you can get the copy from kernel using get_axis_param.

Axis parameter definitions

```
PRA_SSC_SERVO_ABS_CYC_CNT
PRA_SSC_SERVO_ABS_RES_CNT
```

The details of axis parameter please refer to axis parameter table.

Syntax:

```
I32 APS_get_sscnet_servo_abs_position( I32
     Axis_ID, I32 *Cyc_Cnt, I32 *Res_Cnt );
APS_get_sscnet_servo_abs_position( ByVal Axis_ID
     As Long, Cyc_Cnt As Long, Res_Cnt As Long )
     As Long
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535.

`I32 *Cyc_Cnt`: Cycle counter of servo driver

`I32 *Res_Cnt`: Resolution counter of servo driver.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
//1. Initial card and start SSCNET network
//2. Perform a home return operation
Ret = APS_get_sscnet_servo_abs_position(
      Axis_ID, Cyc_Cnt, Res_Cnt );
// Record the abs. position data for next homing
      operation.
```

See also:

```
APS_save_sscnet_servo_abs_position()
APS_load_sscnet_servo_abs_position()
APS_set_axis_param()
APS_get_axis_param()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.4.9 APS_save_sscnet_servo_abs_position

Save absolute reference position to flash ROM

Description:

This function is used to save absolute driver position from axis parameter (SRAM) to flash memory on the controller card.

When system (Controller) is power on, The driver's absolute position will be initialized to zero. After SSCNET network is established, Using "**APS_get_sscnet_servo_abs_position**" function to get the absolute position from driver. SRAM will get a copy at same time. Then, using this function to save all axes absolute position data to flash.

Servo parameters of all axes (16 axes) will be saved at once when you issue this function. You cannot save each servo driver separately.

Syntax:

```
I32 APS_save_sscnet_servo_abs_position( I32
     Board_ID );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

Return Values:

I32 error codes: Please refer to the error code table on page 253.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
//1. Initial card and start SSCNET network
//2. Perform home return operations.
//3. Get abs position for servo drivers.
for(Axis_ID = 0; Axis_ID < 16; Axis_ID ++ )
{
Ret = APS_get_sscnet_servo_abs_position(
      Axis_ID, Cyc_Cnt, Res_Cnt );
}
Ret = APS_save_sscnet_servo_abs_position(
      Board_ID ); //Save all abs. position to
      flash memory.
…
```

See also:

APS_get_sscnet_servo_abs_position()
APS_load_sscnet_servo_abs_position()
APS_set_axis_param()
APS_get_axis_param()

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.4.10 APS_load_sscnet_servo_abs_position

Load absolute reference position from flash ROM

Description:

This function is used to load servo absolute position from flash memory. If user has never saved servo absolute position, calling this function will return error.

User can load all abs. position at once by specified function perameter “Option”. Refer to parameter description.

Syntax:

```
I32 APS_load_sscnet_servo_abs_position( I32
     Axis_ID, I32 Option, I32 *Cyc_Cnt, I32
     *Res_Cnt );
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535

`I32 Option`: Load option.

0: Load abs. position to the axis parameter (SRAM).

1: Load all axes(16 axes) abs. positions to axes parameters.

`I32 *Cyc_Cnt`: Get cycle counter from flash memory. Set this parameter 0 to ignore.

`I32 *Res_Cnt`: Get resolution counter from flash memory. Set this parameter 0 to ingnor.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
//1. Initial card
//2. load abs. position from flash memory.
Ret = APS_load_sscnet_servo_abs_position(Axis_ID,
      1, 0 ,0); //Option = 1 load all axes
APS_set_axis_param( Axis_ID,
      PRA_SSC_SERVO_ABS_POS_OPT, 1 ); //Enable
      abs. position system.
APS_start_sscnet( Board_ID, &AxisFound_InBits );
      //Start SSCNET network.
// Return home position by absolute move
      function.
```

See also:

```
APS_get_sscnet_servo_abs_position()
APS_save_sscnet_servo_abs_position()
APS_set_axis_param()
APS_get_axis_param()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.5 Motion IO and motion status

6.5.1 APS_motion_status

Return motion status

Description:

This function is used to get one axis' motion status. The status includes running, normal stop, abnormal stop by reasons, in waiting other axis, follow status, in some modes, in accelerating or decelerating and so on. Status can be more than two such like mode and running. You need to use this function to check whether the 'Fire-and-forget' function is done in polling system. In even driven system, users can use interrupt event functions.

Please refer to the motion status table for detail description.

Syntax:

```
I32 APS_motion_status( I32 Axis_ID );
APS_motion_status (ByVal Axis_ID As Long) As Long
```

Parameters:

`I32 Axis_ID`:The Axis ID from 0 to 65535

Return Values:

Positive value:

The value motion status. Please refer to motion status bit number definition table for the value meaning

Negative value:

Error Code: Please refers to error code table.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
I32 MotionStatus;
MotionStatus = APS_motion_status( Axis_ID ); //
     Get Motion status
…
```

See also:

```
APS_motion_io_status();
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.5.2 APS_motion_io_status

Return motion IO status

Description:

This function is used to get one axis' motion I/O information like ORG, PEL, MEL, SVON, INP and so on. These statuses are connected to external switched or servo drivers.

Please refer to the motion IO status table for detail description.

Syntax:

```
I32 APS_motion_io_status( I32 Axis_ID );
APS_motion_io_status (ByVal Axis_ID As Long) As
     Long
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535

Return Values:

Positive value:

The value motion IO status. Please refer to motion IO status bit number definition table for the value meaning

Negative value:

Error Code: Please refers to error code table.

Example:

```
I32 MotionIO;
MotionIO = APS_motion_io_status(Axis_ID ); //Get
     Motion IO status
…
```

See also:

```
APS_motion_status ();
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.5.3 APS_set_servo_on

Set servo ON/OFF

Description:

This function is used to command servo driver of specified axis to starts controlling its servomotor. Then motion function could be applied on this axis.

Syntax:

```
I32 APS_set_servo_on( I32 Axis_ID, I32 Servo_on
    );
APS_set_servo_on (ByVal Axis_ID As Long, ByVal
    ServoOn As Long) As Long
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535

`I32 Servo_on`:

0: Servo OFF, 1: Servo ON

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

```
…//Initialization
APS_set_servo_on( Axis_ID, 1 ); // Set servo ON
… //Motion action
APS_set_servo_on(Axis_ID, 0); //Set servo OFF
…//Release
```

See also:

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.5.4 APS_get_position

Get feedback position

Description:

This function is used to get the position information of one axis. The information is in unit of pulse feedback from servo driver or Encoder.

Syntax:

```
I32 APS_get_position( I32 Axis_ID, I32 *Position
     );
APS_get_position (ByVal Axis_ID As Long, Position
     As Long) As Long
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535.

`I32 *Position`: Feedback position. Unit in pulse

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

```
I32 Position;
APS_get_position(Axis_ID, &Position ); //Get
     feedback position
…
```

See also:

```
APS_get_command()
APS_set_position()
APS_set_command()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.5.5 APS_set_position

Set feedback position

Description:

This function is used to set the position information of one axis. The information is in unit of pulse. The 'set' function will change current position or command to a new one. Notice that the set command may cause axis to move violently in closed loop control or absolute control system. But it is no problem in open loop system. The set position command is legal in closed loop or absolute control system because it will also set command to the same value at the same time.

Syntax:

```
I32 APS_set_position(I32 Axis_ID, I32 Position);
APS_set_position (ByVal Axis_ID As Long, ByVal
     Position As Long) As Long
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535.

`I32 Position`: Set feedback position. Unit in pulse.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

```
…
APS_set_position(Axis_ID, 0 ); // Set feedback
     position to zero
```

See also:

```
APS_get_position()
APS_get_command()
APS_set_command()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.5.6 APS_get_command

Get command position

Description:

This function is used to get the command position information of one axis. The information is in unit of pulse.

Syntax:

```
I32 APS_get_command( I32 Axis_ID, I32 *Command );
APS_get_command (ByVal Axis_ID As Long, Command
     As Long) As Long
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535.

`I32 *Command`: Command position. Unit in pulse.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

```
I32 Command ;
APS_get_command(Axis_ID, &Command ); //Get
     command position.
…//
```

See also:

```
APS_get_position()
APS_set_position()
APS_set_command()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.5.7 APS_set_command

Set command position

Description:

This function is used to set the command information of one axis. The information is in unit of pulse. The 'set' function will change current position or command to a new one. Notice that the set command may cause axis to move violently in closed loop control or absolute control system. But it is no problem in open loop system. The set position command is legal in closed loop or absolute control system because it will also set command to the same value at the same time.

Syntax:

```
I32 APS_set_command(I32 Axis_ID, I32 Command);
APS_set_command (ByVal Axis_ID As Long, ByVal
     Command As Long) As Long
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535.

`I32 Command`: Position command. Unit in pulse.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

```
…//
APS_set_command(Axis_ID, 0); //Set command
     position to zero.
```

See also:

```
APS_get_position()
APS_get_command()
APS_set_position()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.6 Single axis motion

6.6.1 APS_relative_move

Begin a relative distance move

Description:

This function is used to start a single axis relative motion. Although there is maximum speed setting in function parameter, the traveling distance and accelerating rate may not be enough due to user's setting to reach the maximum speed. The speed profile's acceleration and deceleration rate and curve are set by axis parameter function. The speed profile's jerk is also set by axis parameter function.

This function is 'fire-and-forget' type. That means user's program or procedure will not be pended during axis traveling. Users must use motion status checking function or interrupt event waiting function to wait it done.

During the axis traveling, users can start a new move command including stop command to override the previous one. The axis will be switched to new command immediately according to new setting of target position, new speed and new jerk.

This command can't be overridden by other motion modes like Jog, home, manual pulse generation, contour motion and G-code program. You must stop axis motion before switching to those modes mentioned above.

Syntax:

```
I32 APS_relative_move( I32 Axis_ID, I32 Distance,
      I32 Max_Speed );
APS_relative_move (ByVal Axis_ID As Long, ByVal
      Distance As Long, ByVal Max_Speed As Long)
      As Long
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 Distance: Relative distance. Unit in pulse.

I32 Max_Speed: The maximum speed of this move profile. Unit: pulse/sec.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

See also:

```
APS_relative_move()
APS_absolute_move()
APS_velocity_move()
APS_home_move()
APS_stop_move()
APS_emg_stop();
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.6.2 APS_absolute_move

Begin a absolute position move

Description:

This function is used to start a single axis absolute positioning motion. Although there is maximum speed setting in function parameter, the traveling distance and accelerating rate may not be enough due to user's setting to reach the maximum speed. The speed profile's acceleration and deceleration rate and curve are set by axis parameter function. The speed profile's jerk is also set by axis parameter function.

This function is 'fire-and-forget' type. That means user's program or procedure will not be pended during axis traveling. Users must use motion status checking function or interrupt event waiting function to wait it done.

During the axis traveling, users can start a new move command including stop command to override the previous one. The axis will be switched to new command immediately according to new setting of target position, new speed and new jerk.

This command can't be overridden by other motion modes like Jog, home, manual pulse generation, contour motion and G-code program. Users must stop axis motion before switching to those modes mentioned above.

Syntax:

```
I32 APS_absolute_move( I32 Axis_ID, I32 Position,
     I32 Max_Speed );
APS_absolute_move (ByVal Axis_ID As Long, ByVal
     Position As Long, ByVal Max_Speed As Long)
     As Long
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 Position: Absolute command position. Unit in pulse.

I32 Max_Speed: The maximum speed of this move profile. Unit: pulse/sec

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

See also:

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.6.3 APS_velocity_move

Begin a velocity move

Description:

This function is used to start a velocity move. The axis will stop when users issue stop move command. The speed profile's acceleration and deceleration rate and curve are set by axis parameter function. The speed profile's jerk is also set by axis parameter function.

This function is 'fire-and-forget' type. That means user's program or procedure will not be pended during axis traveling. Users must use motion status checking function or interrupt event waiting function to wait it done.

During the axis traveling, users can start a new move command including stop command to override the previous one. The axis will be switched to new command immediately according to new setting of target position, new speed and new jerk.

This command can't be overridden by other motion modes like Jog, home, manual pulse generation, contour motion and G-code program. Users must stop axis motion before switching to those modes mentioned above.

Syntax:

```
I32 APS_velocity_move( I32 Axis_ID, I32 Max_Speed
    );
APS_velocity_move (ByVal Axis_ID As Long, ByVal
    Max_Speed As Long) As Long
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535.

`I32 Max_Speed`: The maximum speed of this move profile. Unit : pulse/sec

Return Values:

I32 Error code: Please refer to error code table.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
APS_set_axis_param(Axis_ID, PRA_ACC, 1000000 ); /
     /Set acceleration rate
APS_set_axis_param(Axis_ID, PRA_DEC, 1000000 ); /
     /Set deceleration rate
APS_velocity_move(Axis_ID, Max_Speed ); //Start
     velocity move
…
APS_stop_move(Axis_ID); //Stop velocity move
```

See also:

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.6.4 APS_home_move

Begin a home move

Description:

This function is used to start a HOME(ORG or DOG) position of the axis. There are several modes which can be selected by axis parameter setting functions. After it is done, the position of the axis will be renew base on the physical location of HOME.

This function is 'fire-and-forget' type. That means user's program or procedure will not be pended during axis traveling. Users must use motion status checking function or interrupt event waiting function to wait it done.

Syntax:

```
I32 APS_home_move( I32 Axis_ID );
APS_home_move (ByVal Axis_ID As Long) As Long
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
//Set homing parameters
APS_set_axis_param( Axis_ID, PRA_HOME_MODE, 0 );
     //Set home mode
APS_set_axis_param( Axis_ID, PRA_HOME_DIR, 1 ); /
     /Set home direction
APS_set_axis_param( Axis_ID, PRA_HOME_CURVE, 0 );
     //Set acceleration paten (T-curve)
APS_set_axis_param( Axis_ID, PRA_HOME_ACC,
     1000000 ); //Set homing acceleration rate
APS_set_axis_param( Axis_ID, PRA_HOME_VS, 0 ); //
     Set homing start velocity
APS_set_axis_param( Axis_ID, PRA_HOME_VM, 2000000
     ); //Set homing maximum velocity.
APS_set_axis_param( Axis_ID, PRA_HOME_VO, 200000
     ); //Set homing

APS_home_move(Axis_ID ); //Start homing
…//Check homing done(Motion done)
```

See also:

```
APS_set_axis_param()
APS_get_axis_param()
APS_stop_move()
APS_emg_stop()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.6.5 APS_set_jog_param

Set jog parameters

Description:

This function is used to set jog move relative parameters.

Syntax:

```
I32 APS_set_jog_param( I32 Axis_ID, JOG_DATA
     *pStr_Jog, I32 Mask );
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535.

`JOG_DATA *pStr_Jog`: Structure of jog move parameters. Define in “type_def.h”

```
typedef struct
{
     I16 i16_jogMode; // Jog mode. 0:Free running
     mode, 1:Step mode
     I16 i16_dir;// Jog direction. 0:positive,
     1:negative direction
     I16 i16_accType;// Acceleration and
     Deceleration pattern 0: T-curve, 1: S-curve
     I32 i32_acc;// Acceleration rate ( pulse /
     s2 )
     I32 i32_dec;// Deceleration rate ( pulse /
     s2 )
     I32 i32_maxSpeed; // A Positive value,
     maximum velocity. ( pulse / s )
     I32 i32_offset;  // A Positive value, step
     offset. For step jog mode. (pulse)
     I32 i32_delayTime; // Delay time, For step
     jog mode. ( range: 0 ~ 65535 millisecond,
     align by cycle time)
} JOG_DATA;
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

I32 Mask: Mask parameter setting. Bit format, set 0 will be masked.

Mask item	Mask bit number
Acceleration rate (i32_acc)	0
Deceleration rate(i32_dec)	1
Maximum velocity (i32_maxSpeed)	2
Step offset (i32_offset)	3
Delay time (i32_delayTime)	4
Jog mode (i16_jog mode)	5
Jog direction (i16_direction)	6
Jog acceleration/deceleration pattern	7

Return Values:

I32 Error code: Refer to error code table.

Example:

```
#include "type_def.h"
#include "APS168.h"
// Initial cards first…

I32 ret;
JOG_DATA jog;

jog.i16_jogMode = 1; //Mask = 0x20
jog.i16_dir = 0; //Mask = 0x40

ret = APS_set_jog_param( Axis_ID, &jog, 0x20 |
      0x40 );
if( ret != 0 ) //Error
```

See also:

```
APS_set_axis_param()
APS_get_axis_param()
APS_get_jog_param()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.6.6 APS_get_jog_param

Get jog parameters

Description:

This function is used to get jog move relative parameters.

Syntax:

```
I32 APS_get_jog_param( I32 Axis_ID, JOG_DATA
    *pStr_Jog );
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535.

`JOG_DATA *pStr_Jog`: Structure of jog move parameters. Define in "type_def.h"

```
typedef struct
{
    I16 i16_jogMode; // Jog mode. 0:Free running
    mode, 1:Step mode
    I16 i16_dir;// Jog direction. 0:positive,
    1:negative direction
    I16 i16_accType;// Acceleration and
    Deceleration pattern 0: T-curve, 1: S-curve
    I32 i32_acc;// Acceleration rate ( pulse /
    s2 )
    I32 i32_dec;// Deceleration rate ( pulse /
    s2 )
    I32 i32_maxSpeed; // A Positive value,
    maximum velocity. ( pulse / s )
    I32 i32_offset;  // A Positive value, step
    offset. For step jog mode. (pulse)
    I32 i32_delayTime; // Delay time, For step
    jog mode. ( range: 0 ~ 65535 millisecond,
    align by cycle time)
} JOG_DATA;
```

Return Values:

I32 error codes: Please refer to the error code table on page 253.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
#include “type_def.h”
#include “APS168.h”
// Initial cards first…

I32 ret;
JOG_DATA jog;

ret = APS_get_jog_param( Axis_ID, &jog );
if( ret != 0 ) //Error
```

See also:

APS_set_axis_param()
APS_get_axis_param()
APS_set_jog_param()

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.6.7 APS_jog_mode_switch

Enable / Disable jog move

Description:

This function is used to switch specified axis to jog mode. When the axis is in jog mode, it cannot accept other move command except stop command.

Users must enable jog move mode before perform jog move.

Syntax:

```
I32 APS_jog_mode_switch( I32 Axis_ID, I32 Turn_No
    );
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535.

`I32 Turn_No`: 0:Disable jog mode, 1:Enable jog mode.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

```
// Configure jog move parameter.

ret = APS_jog_mode_switch(Axis_ID, 1 ); //Turn on
    jog move mode.
// perform jog move …(APS_jog_start)
…
ret = APS_jog_mode_switch(Axis_ID, 0 ); //Turn
    off jog move mode.
// perform other move commands
```

See also:

```
APS_set_jog_param()
APS_get_jog_param()
APS_jog_start()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.6.8 APS_jog_start

Start / stop jog move

Description:

This function is used to start / stop a jog move. Before start a jog move, you must enable the axis to jog mode.

Syntax:

```
I32 APS_jog_start( I32 Axis_ID, I32 STA_On );
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 STA_On: 1:STA signal on, 0:STA signal off.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

```
// Configure jog move parameter.

ret = APS_jog_mode_switch(Axis_ID, 1 ); //Turn on
      jog move mode.

// perform jog move …(APS_jog_start)
APS_jog_start( Axis_ID,1 ); //STA signal ON
…
APS_jog_start(Axis_ID, 0); //STA signal OFF

ret = APS_jog_mode_switch(Axis_ID, 0 ); //Turn
      off jog move mode.
// perform other move commands
```

See also:

```
APS_set_jog_param()
APS_get_jog_param()
APS_jog_mode_switch();
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.6.9 APS_stop_move

Stop move

Description:

This function is used to stop single or multiple axes motion at once. It can stop single axis homing, positioning and speed moving. It also can stop multiple axes interpolation motion when users place one of axis ID which is relative to interpolation moving. The deceleration profile is set by axis parameter function.

The stop function can't be overridden by other functions.

Syntax:

```
I32 APS_stop_move(I32 Axis_ID);
APS_stop_move (ByVal Axis_ID As Long) As Long
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

```
// APS_absolute_move(Axis_ID, Position, Max_Speed
     );
// APS_home_move(Axis_ID ); //Home move
…
APS_stop_move(Axis_ID); //Stop move
```

See also:

```
APS_emg_stop()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.6.10 APS_emg_stop

Emergency stop

Description:

This function is used to stop single or multiple axes motion immediately. It can stop single axis homing, positioning and speed moving. It also can stop multiple axes interpolation motion when users place one of axis ID which is relative to interpolation moving. Because the stop function will stop axis accidentally, it will generate an abnormal stop interrupt event rather than normal stop event if interrupt factor is set. The motion status will also be set to an abnormal stop status. The abnormal stop status or event will be clear by next motion command.

Syntax:

```
I32 APS_emg_stop(I32 Axis_ID);
APS_emg_stop (ByVal Axis_ID As Long) As Long
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535.

Return Values:

`I32 Error code`: Refer to error code table.

Example:

```
// APS_absolute_move(Axis_ID, Position, Max_Speed
     );
// APS_home_move(Axis_ID ); //Home move
…
APS_emg_stop (Axis_ID); //EMG stop
```

See also:

```
APS_stop_move()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.7 Interpolation

6.7.1 APS_absolute_linear_move

Begin a absolute position linear interpolation

Description:

This function is used to start an absolute linear interpolation positioning motion. Although there is maximum speed setting in function parameter, the traveling distance and accelerating rate may not be enough due to user's setting to reach the maximum speed. The speed profile's acceleration and deceleration rate and curve are set by axis parameter function. The speed profile's jerk is also set by axis parameter function. Because the speed parameter is in vector direction, this function will take the master axis's acceleration and deceleration time constant to calculate. The master axis is the minor axis number that user perform a interpolation.

This function is 'fire-and-forget' type. That means user's program or procedure will not be pended during axis traveling. Users must use motion status checking function or interrupt event waiting function to wait it done.

During the axis traveling, users can start a new move command including stop command to override the previous one. The axis will be switched to new command immediately according to new setting of target position, new speed and new jerk.

The overridden command must have the same dimension and axis ID of previous one. These two commands can't be overridden by other motion modes like home operation. Users must stop axis motion before switching to those modes mentioned above.

Note: The axes specified in Axis_ID_Array must be of the same card.

Syntax:

```
I32 APS_absolute_linear_move( I32 Dimension, I32
     *Axis_ID_Array, I32 *Position_Array, I32
     Max_Linear_Speed );
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Parameters:

I32 Dimension: The dimension of interpolation axes. (2~4 axes)

I32 *Axis_ID_Array: The axis ID array from 0 to 65535.

I32 *Position_Array: Absolute position array. (unit: pulse)

I32 Max_Linear_Speed: Maximum linear interpolation speed (unit: pulse/sec)

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

```
//…Initial card
I32 Dimension = 4;
I32 Master_Axis_ID = 1; //Master axis
I32 Axis_ID_Array[4] = { 1, 2, 3, 4}; //Axis ID 1
     is master axis.
I32 Position_Array [4] = {10000, 20000, 30000,
     40000 };
I32 Max_Linear_Speed = 10000;
I32 Ret;
APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 0
     ); //Set T-curve
APS_set_axis_param( Master_Axis_ID, PRA_ACC,
     100000 ); //Set acceleration
APS_set_axis_param( Master_Axis_ID, PRA_DEC,
     100000 ); //Set deceleration

Ret = APS_absolute_linear_move ( Dimension,
     Axis_ID_Array, Position_Array,
     Max_Linear_Speed );
…
```

See also:

APS_relative_linear_move

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.7.2 APS_relative_linear_move

Begin a relative distance linear interpolation

Description:

This function is used to start a relative linear interpolation positioning motion. Although there is maximum speed setting in function parameter, the traveling distance and accelerating rate may not be enough due to user's setting to reach the maximum speed. The speed profile's acceleration and deceleration rate and curve are set by axis parameter function. The speed profile's jerk is also set by axis parameter function. Because the speed parameter is in vector direction, this function will take the master axis's acceleration and deceleration time constant to calculate. The master axis is the minor axis number that user perform an interpolation.

This function is 'fire-and-forget' type. That means user's program or procedure will not be pended during axis traveling. Users must use motion status checking function or interrupt event waiting function to wait it done.

During the axis traveling, users can start a new move command including stop command to override the previous one. The axis will be switched to new command immediately according to new setting of target position, new speed and new jerk.

The overridden command must have the same dimension and axis ID of previous one. These two commands can't be overridden by other motion modes like home operation. Users must stop axis motion before switching to those modes mentioned above.

Note: The axes specified in Axis_ID_Array must be of the same card.

Syntax:

```
I32 APS_relative_linear_move( I32 Dimension, I32
     *Axis_ID_Array, I32 *Distance_Array, I32
     Max_Linear_Speed );
APS_relative_linear_move( ByVal Dimension As
     Long,  Axis_ID_Array As Long, Distance_Array
     As Long, ByVal Max_Linear_Speed As Long) As
     Long
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Parameters:

I32 Dimension: The dimension of interpolation axes. (2~4 axes)

I32 *Axis_ID_Array: The Axis ID array from 0 to 65535.

I32 *Distance_Array: Relative distance array. (unit: pulse)

I32 Max_Linear_Speed: Maximum linear interpolation speed (unit: pulse/sec)

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

```
//…Initial card
I32 Dimension = 4;
I32 Master_Axis_ID = 0;
I32 Axis_ID_Array[4] = {0, 1, 2, 3}; //Axis ID 0
     is master axis.
I32 Distance_Array[4] = {10000, 20000, 30000,
     40000 };
I32 Max_Linear_Speed = 10000;
I32 Ret;
APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 1
     ); //Set S-curve
APS_set_axis_param( Master_Axis_ID, PRA_ACC,
     100000 ); //Set acceleration
APS_set_axis_param( Master_Axis_ID, PRA_DEC,
     100000 ); //Set deceleration

Ret = APS_relative_linear_move( Dimension,
     Axis_ID_Array, Distance_Array,
     Max_Linear_Speed );
…
```

See also:

```
APS_relative_linear_move()
APS_set_axis_param()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.7.3 APS_absolute_arc_move

Begin an absolute position circular interpolation

Description:

This function is used to start an absolute two dimension circular interpolation positioning motion. User must specify absolute center position and traveling angle for circular interpolation. The speed profile's acceleration and deceleration rate are set by axis parameter function. The speed profile's jerk is also set by axis parameter function. The following axis parameter should be setting before you calling this function.

```
PRA_CURVE
PRA_ACC
PRA_DEC
PRA_VS
PRA_VE
```

The details of parameters please refer the axis parameter table.

Although there is maximum speed setting in function parameter, the traveling distance and accelerating rate may not be enough due to user's setting to reach the maximum speed. Because the speed parameter is in vector direction(Tangent to the circular), this function will take the master axis's acceleration and deceleration time constant to calculate. The master axis is the minor axis number that user perform an interpolation. For example, Axis ID 2 and 3 are performing a circular interpolation. The Axis ID 2 is the master axis.

This function is 'fire-and-forget' type. That means user's program or procedure will not be pended during axis traveling. Users must use motion status checking function or interrupt event waiting function to wait it done. The motion status "circular interpolation signal (CIP)" of each axis performing a circular interpolation will be turn on when command is started and will be turned off at command is finished. If circular interpolation is stop normally, the normal stop signal (NSTP) will be turned on. On the contrary, if circular interpolation is stopped abnormally (such as ALM, EMG, SEMG and so on is turned on), abnormal stop signal (ASTP) will be turned on.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

During the axis traveling, users can start a new move command including stop command to override the previous one (The dimension and Axis_ID_Array must be the same). The axis will be switched to new command immediately according to new setting of target center position, new speed profile and new jerk.

This command can't be overridden by other motion modes like home operation. Users must stop axis motion before switching to those modes mentioned above.

Note: The 2 axes specified in Axis_ID_Array must be of the same card.

Syntax:

```
I32 APS_absolute_arc_move( I32 Dimension, I32
     *Axis_ID_Array, I32 *Center_Pos_Array, I32
     Max_Arc_Speed, I32 Angle );
```

Parameters:

`I32 Dimension`: The dimension of interpolation axes. (Only support 2 dimension)

`I32 *Axis_ID_Array`: The Axis ID array from 0 to 65535.

`I32 *Center_Pos_Array`: Absolute circular center position. Unit: pulse.

`I32 Max_Arc_Speed`: Maximum circular interpolation speed (Circular tangent speed). Unit: pulse/sec

`I32 Angle`: Travel angle. Value range: -360 to 360 degree. Positive for counterclockwise.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
I32 Dimension = 2; //2 Dimension only
I32 Axis_ID_Array[2] = { 2, 4 }; //Axis_ID 2 is
     the master axis.
I32 Master_Axis_ID = 2; //Axis_ID 2 is the master
     axis.
I32 Center_Pos_Array[2] = {100000, 0};
I32 Max_Arc_Speed = 10000; // pulse/sec
I32 Angle = -180; // clockwise 180 degree.
I32 Ret; //Return code.

//…
APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 1
     ); //Set S-curve
APS_set_axis_param( Master_Axis_ID, PRA_ACC,
     100000 ); //Set acceleration
APS_set_axis_param( Master_Axis_ID, PRA_DEC,
     100000 ); //Set deceleration
Ret = APS_absolute_arc_move( Dimension,
     Axis_ID_Array, Center_Pos_Array,
     Max_Arc_Speed, Angle ); //Perform a circular
     interpolation
```

See also:

```
APS_relative_arc_move()
APS_set_axis_param ()
APS_get_axis_param ()
APS_motion_status()
APS_stop_move()
APS_emg_stop()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.7.4 APS_relative_arc_move

Begin a relative distance circular interpolation

Description:

This function is used to start an relative two dimension circular interpolation positioning motion. User must specified a center position relative current commend position and traveling angle for circular interpolation. The speed profile's acceleration and deceleration rate and curve are set by axis parameter function. The speed profile's jerk is also set by axis parameter function. The following axis parameter should be setting before you calling this function.

```
PRA_CURVE
PRA_ACC
PRA_DEC
PRA_VS
PRA_VE
```

The details of parameters please refer the axis parameter table.

Although there is maximum speed setting in function parameter, the traveling distance and accelerating rate may not be enough due to user's setting to reach the maximum speed. Because the speed parameter is in vector direction (tangent to the circular), this function will take the master axis's acceleration and deceleration time constant to calculate. The master axis is the minor axis number that user perform an interpolation. For example, Axis ID 2 and 3 are performing a circular interpolation. Therefore, the Axis ID 2 is the master axis.

This function is 'fire-and-forget' type. That means user's program or procedure will not be pended during axis traveling. Users must use motion status checking function or interrupt event waiting function to wait it done. Motion status: in circular interpolation signal (CIP) will be turn on when it start and will be turn off at command is finished. If circular interpolation is stop normally, the normal stop signal (NSTP) will be turn on. On the contrary, circular interpolation is stopped abnormally (ALM, EMG, SEMG, and so on), abnormal stop signal (ASTP) will be turn on.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

During the axis traveling, users can start a new move command including stop command to override the previous one (The dimension and Axis_ID_Array must be the same). The axis will be switched to new command immediately according to new setting of target center position, new speed profile and new jerk.

This command can't be overridden by other motion modes like home operation. Users must stop axis motion before switching to those modes mentioned above.

Note: The 2 axes specified in Axis_ID_Array must be of the same card.

Syntax:

```
I32 APS_relative_arc_move( I32 Dimension, I32
     *Axis_ID_Array, I32 *Center_Offset_Array,
     I32 Max_Arc_Speed, I32 Angle );
```

Parameters:

`I32 Dimension`: The dimension of interpolation axes. (Only support 2 dimension)

`I32 *Axis_ID_Array`: The Axis ID array from 0 to 65535.

`I32 *Center_Offset_Array`: circular center position relative to current command position. Unit: pulse

`I32 Max_Arc_Speed`: Maximum circular interpolation speed (Circular tangent speed). Unit: pulse/sec

`I32 Angle`: Travel angle. Value range: -360 to 360 degree. Positive for counterclockwise.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
I32 Dimension = 2; //2 Dimension only
I32 Axis_ID_Array[2] = { 1, 3}; //Axis_ID 1 is
     the master axis.
I32 Master_Axis_ID = 1; //Axis_ID 1 is the master
     axis.
I32 Center_Offset_Array [2] = {300000, 0};
I32 Max_Arc_Speed = 20000; // pulse/sec
I32 Angle = 90; // counterclockwise 90 degree.
I32 Ret; //Return code.

//…
APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 1
     ); //Set S-curve
APS_set_axis_param( Master_Axis_ID, PRA_ACC,
     100000 ); //Set acceleration
APS_set_axis_param( Master_Axis_ID, PRA_DEC,
     100000 ); //Set deceleration
Ret = APS_relative_arc_move( Dimension,
     Axis_ID_Array, Center_Offset_Array,
     Max_Arc_Speed, Angle ); //Perform a circular
     interpolation
```

See also:

```
APS_absolute_arc_move ()
APS_set_axis_param ()
APS_get_axis_param ()
APS_motion_status()
APS_stop_move()
APS_emg_stop()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.8 Interrupt

6.8.1 APS_int_enable

Interrupt main switch

Description:

This function is used to enable/disable interrupt of one board to host computer. It is a hardware switch for this board. Once it is disabled, host computer will not received any hardware interrupt even the interrupt factor is enabled on the board. Users must enable this function before using any interrupt relative functions and disable this function when users do not use interrupt anymore.

Syntax:

```
I32 APS_int_enable( I32 Board_ID, I32 Enable );
APS_int_enable (ByVal Board_ID As Long, ByVal
     Enable As Long) As Long
```

Parameters:

`I32 Board_ID`: ID of the target controller. It's retrieved by successful call to APS_initial().

`I32 Enable`: Enable/Disable interrupt.

0: Disable. 1: Enable

Return Values:

I32 error codes: Please refer to the error code table on page 253.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
I32 Int_No; //Interrupt number
I32 returnCode; // function return code

Int_No = APS_set_int_factor( Board_ID, Item_No,
     Factor_No, 1 ); //Enable the interrupt
     factor
APS_int_enable( Board_ID, 1 ); //Enable the
     interrupt main switch
returnCode = APS_wait_single_int( Int_No,
     Time_Out ); //Wait interrupt
if( returnCode == ERR_NoError )
{ //Interrupt occurred
APS_reset_int( Int_No );
     …//Do something
}

APS_set_int_factor( Board_ID, Item_No, Factor_No,
     0 ); //Disable the interrupt factor
APS_int_enable( Board_ID, 0 ); //Disable the
     interrupt main switch
```

See also:

```
APS_set_int_factor()
APS_get_int_factor()
APS_wait_single_int()
APS_wait_multiple_int()
APS_reset_int()
APS_set_int()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.8.2 APS_set_int_factor

Enable/Disable interrupt factor and get interrupt handle.

Description:

This function is used to turn on/off the interrupt factor bit. If it is turned on, the function will return a notification event for this bit and return an I32 type event number. Users can wait this event by assigning corresponding event number into a wait function. The event number is unique in one system.

The interrupt factor definition, please refer to the interrupt factor table.

Syntax:

```
I32 APS_set_int_factor( I32 Board_ID, I32
     Item_No, I32 Factor_No, I32 Enable );
APS_set_int_factor (ByVal Board_ID As Long, ByVal
     Item_No As Long, ByVal Factor_No As Long,
     ByVal Enable As Long) As Long
```

Parameters:

`I32 Board_ID`: ID of the target controller. It's retrieved by successful call to APS_initial().

`I32 Item_No`: Interrupt factor table item number. Refer to interrupt factor table.

`I32 Factor_No`: Factor number of one item. Refer to interrupt factor table.

`I32 Enable`: Enable interrupt factor

0: Disable

1:Enable

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Return Values:

When:

[Enable = 1] : Enable the interrupt factor

Return positive value: I32 Interrupt event number.

Return negative value: I32 error code. Refer to error code table.

[Enable = 0] : Disable the interrupt factor

Return I32 error code. Refer to error code table.

Example:

```
I32 Int_No; //Interrupt number
I32 returnCode; // function return code

Int_No = APS_set_int_factor( Board_ID, Item_No,
     Factor_No, 1 ); //Enable the interrupt
     factor
APS_int_enable( Board_ID, 1 ); //Enable the
     interrupt main switch
returnCode = APS_wait_single_int( Int_No,
     Time_Out ); //Wait interrupt
if( returnCode == ERR_NoError )
{ //Interrupt occurred
     APS_reset_int( Int_No );
     …//Do something
}

APS_set_int_factor( Board_ID, Item_No, Factor_No,
     0 ); //Disable the interrupt factor
APS_int_enable( Board_ID, 0 ); //Disable the
     interrupt main switch
```

See also:

```
APS_int_enable()
APS_get_int_factor()
APS_wait_single_int()
APS_wait_multiple_int()
APS_reset_int()
APS_set_int()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.8.3 APS_get_int_factor

Get interrupt factor enable or disable

Description:

This function is used to get the setting of interrupt factor.

Syntax:

```
I32 APS_get_int_factor( I32 Board_ID, I32
     Item_No, I32 Factor_No, I32 *Enable );
APS_get_int_factor (ByVal Board_ID As Long, ByVal
     Item_No As Long, ByVal Factor_No As Long,
     Enable As Long) As Long
```

Parameters:

`I32 Board_ID`: ID of the target controller. It's retrieved by successful call to APS_initial().

`I32 Item_No`: Interrupt factor table item number. Refer to interrupt factor table.

`I32 Factor_No`: Factor number of one item. Refer to interrupt factor table.

`I32 *Enable`: Return enable or disable. 0: Disable, 1:Enable.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
I32 ReturnCode;
I32 Enable;
ReturnCode = APS_get_int_factor( Board_ID,
     Item_No, Factor_No, &Enable );
…
```

See also:

```
APS_int_enable()
APS_set_int_factor()
APS_wait_single_int()
APS_wait_multiple_int()
APS_reset_int()
APS_set_int();
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.8.4 APS_wait_single_int

Wait single interrupt event

Description:

When the user enabled the interrupt function for specified factors by "**APS_set_int_factor**", it could use this function to wait a specific interrupt. When this function was running, the process would never stop until the event was be triggered or the function was time out. This function returns when one of the following occurs:

1. The specified interrupt factor is in the signaled state.
2. The time-out interval elapses.

This function checks the current state of the specified interrupt factor. If the state is non-signaled, the calling thread enters the wait state. It uses no processor time while waiting for the INT state to become signaled or the time-out interval to elapse.

When the interrupt is occurred and the wait function is return. User should use **APS_reset_int** () to reset the interrupt by themselves. If user does not reset the interrupt, the wait function will pass immediately next time.

Syntax:

```
I32 APS_wait_single_int( I32 Int_No, I32 Time_Out
    );
APS_wait_single_int (ByVal Int_No As Long, ByVal
    Time_Out As Long) As Long
```

Parameters:

I32 Int_No: Interrupt event number. Get from APS_set_int_factor() function.

I32 Time_Out: Wait timeout time. Unit in mini-second. If value is set -1, the function's time-out interval never elapses (infinite). If Time_Out is zero, the function tests the interrupt's state and returns immediately.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Return Values:

ERR_NoError(0): The event is wait success.

I32 error codes: Please refer to the error code table on page 253.

Example:

```
I32 Int_No; //Interrupt number
I32 returnCode; // function return code

Int_No = APS_set_int_factor( Board_ID, Item_No,
     Factor_No, 1 ); //Enable the interrupt
     factor
APS_int_enable( Board_ID, 1 ); //Enable the
     interrupt main switch
returnCode = APS_wait_single_int( Int_No,
     Time_Out ); //Wait interrupt
if( returnCode == ERR_NoError )
{ //Interrupt occurred
     APS_reset_int( Int_No );
     …//Do something
}

APS_set_int_factor( Board_ID, Item_No, Factor_No,
     0 ); //Disable the interrupt factor
APS_int_enable( Board_ID, 0 ); //Disable the
     interrupt main switch
```

See also:

```
APS_int_enable()
APS_set_int_factor()
APS_get_int_factor()
APS_wait_multiple_int()
APS_reset_int()
APS_set_int()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.8.5 APS_wait_multiple_int

Wait multiple interrupt events

Description:

When the user enabled the interrupt function for specified factors by "**APS_set_int_factor**()", users could use this function to wait specific interrupts. When this function was running, the process would never stop until the event was be triggered or the function was time out. This function returns when one of the following occurs:

1. Either any one or all of the interrupt factors are in the signaled state.
2. The time-out interval elapses.

This function checks the current state of the specified interrupt factor. If the state is non-signaled, the calling thread enters the wait state. It uses no processor time while waiting for the INT state to become signaled or the time-out interval to elapse.

Users must use **APS_reset_int**() to reset the events themselves before wait the events next time.

Syntax:

```
I32 APS_wait_multiple_int( I32 Int_Count, I32
     *Int_No_Array, I32 Wait_All, I32 Time_Out );
APS_wait_multiple_int (ByVal Int_Count As Long,
     Int_No_Array As Long, ByVal Wait_All As
     Long, ByVal Time_Out As Long) As Long
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Parameters:

I32 Int_Count: Specifies the number of Interrupt. The maximum number of factors is 64.

I32 *Int_No_Array: Interrupt event number array. Get from APS_set_int_factor() function.

I32 Wait_All: Wait option.

FALSE: (0) The function returns when the state of any one of the events in the array is signaled.

TRUE: (1) The function returns when the state of all events in the array is signaled.

I32 Time_Out: Wait timeout time. Unit in mini-second. If value is set -1, the function's time-out interval never elapses (infinite).

Return Values:

Postive value: (Int_Count – 1): The events are wait success.

If Wait_All is FALSE (0), the return value indicates that the state of all specified objects is signaled.

If WaitAll is FALSE(0), the return value indicates the array index of the object that satisfied the wait. If more than one event became signalled during the call, this is the array index of the signalled object with the smallest index value of all the signalled objects.

Negative value: I32 error code: Refer to error code table.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
I32 Int_No[2]; //Interrupt number
I32 returnCode; // function return code

Int_No[0] = APS_set_int_factor( Board_ID,
     Item_No1, Factor_No1, 1 ); //Enable the
     interrupt factor
Int_No[1] = APS_set_int_factor( Board_ID,
     Item_No2, Factor_No2, 1 ); //Enable the
     interrupt factor

APS_int_enable( Board_ID, 1 ); //Enable the
     interrupt main switch
returnCode = APS_wait_multiple_int( 2, Int_No, 1,
     I32 Time_Out ); //Wait multiple interrupts,
     (wait all)
if( returnCode == ERR_NoError )
{ //Interrupts occurred
     APS_reset_int( Int_No[0] );
APS_reset_int( Int_No[1] );
     …//Do something
}

APS_set_int_factor( Board_ID, Item_No1,
     Factor_No1, 0 ); //Disable the interrupt
     factor
APS_set_int_factor( Board_ID, Item_No2,
     Factor_No2, 0 ); //Disable the interrupt
     factor
APS_int_enable( Board_ID, 0 ); //Disable the
     interrupt main switch
```

See also:

```
APS_int_enable()
APS_set_int_factor()
APS_get_int_factor()
APS_wait_single_int()
APS_reset_int()
APS_set_int()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.8.6 APS_reset_int

Reset interrupt event to non-signaled state.

Description:

This function is used to reset singled event to non-singled state.

Syntax:

```
I32 APS_reset_int( I32 Int_No );
APS_reset_int (ByVal Int_No As Long) As Long
```

Parameters:

I32 Int_No: Interrupt event number. Get from APS_set_int_factor() function.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
I32 Int_No; //Interrupt number
I32 returnCode; // function return code

Int_No = APS_set_int_factor( Board_ID, Item_No,
     Factor_No, 1 ); //Enable the interrupt
     factor
APS_int_enable( Board_ID, 1 ); //Enable the
     interrupt main switch
returnCode = APS_wait_single_int( Int_No,
     Time_Out ); //Wait interrupt
if( returnCode == ERR_NoError )
{ //Interrupt occurred
     APS_reset_int( Int_No );
     …//Do something
}

APS_set_int_factor( Board_ID, Item_No, Factor_No,
     0 ); //Disable the interrupt factor
APS_int_enable( Board_ID, 0 ); //Disable the
     interrupt main switch
```

See also:

```
APS_int_enable()
APS_set_int_factor()
APS_get_int_factor()
APS_wait_single_int()
APS_wait_multiple_int()
APS_set_int()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.8.7 APS_set_int

Set interrupt event to signaled state.

Description:

This function is used to signal the specified event interrupt. The wait function will return (pass) when this function is set.

Syntax:

```
I32 APS_set_int( I32 Int_No );
APS_set_int (ByVal Int_No As Long) As Long
```

Parameters:

I32 Int_No: Interrupt event number. Get from APS_set_int_factor() function.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
I32 Int_No; //Interrupt number
I32 returnCode; // function return code

Int_No = APS_set_int_factor( Board_ID, Item_No,
     Factor_No, 1 ); //Enable the interrupt
     factor
APS_int_enable( Board_ID, 1 ); //Enable the
     interrupt main switch
APS_set_int(Int_No ); //Signaled interrupt event.
returnCode = APS_wait_single_int( Int_No,
     Time_Out ); //Wait function will pass
     immediately
if( returnCode == ERR_NoError )
{ //Interrupt occurred
     APS_reset_int( Int_No );
     …//Do something

APS_set_int_factor( Board_ID, Item_No, Factor_No,
     0 ); //Disable the interrupt factor
APS_int_enable( Board_ID, 0 ); //Disable the
     interrupt main switch
```

See also:

```
APS_int_enable()
APS_set_int_factor()
APS_get_int_factor()
APS_wait_single_int()
APS_wait_multiple_int()
APS_reset_int()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.9 Sampling

6.9.1 APS_set_sampling_param

Set sampling parameter.

Description:

This function is used to set sampling parameters such as sampling rate, sampling channel source and so on. Please refer to the sampling parameters table for the definition and detail descriptions.

Syntax:

```
I32 APS_set_sampling_param( I32 Board_ID, I32
     Param_No, I32 Param_Dat );
APS_set_sampling_param( ByVal Board_ID As Long,
     ByVal ParaNum As Long, ByVal ParaDat As Long
     ) As Long
```

Parameters:

`I32 Board_ID`: ID of the target controller. It's retrieved by successful call to APS_initial().

`I32 Param_No`: Specified sampling parameter number, refer to sampling parameter table for definition.

`I32 Param_Dat`: The corresponding parameter value of sampling number. Refer to the sampling table.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
//... initial card.
I32 Ret = APS_set_sampling_param( Board_ID,
     SAMP_PA_RATE, 2  ); //Set sampling rate
…
```

See also:

```
APS_get_sampling_param()
APS_wait_trigger_sampling()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.9.2 APS_get_sampling_param

Get sampling parameter.

Description:

This function is used to get sampling parameters such as sampling rate, sampling channel source and so on. Please refer to the sampling parameters table for the definition and detail descriptions.

Syntax:

```
I32 APS_get_sampling_param( I32 Board_ID, I32
      ParaNum, I32 *ParaDat );
APS_get_sampling_param( ByVal Board_ID As Long,
      ByVal ParaNum As Long, ParaDat As Long ) As
      Long
```

Parameters:

`I32 Board_ID`: ID of the target controller. It's retrieved by successful call to APS_initial().

`I32 ParaNum`: Sampling parameter number. Refer to the sampling parameter table.

`I32 *ParaDat`: Return sampling parameter value. Refer to the sampling parameter table.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
I32 ParaDat:
Ret = APS_set_sampling_param( Board_ID,
     SAMP_PA_EDGE, & ParaDat  ); //Get trigger
     edge
…
```

See also:

```
APS_set_sampling_param()
APS_wait_trigger_sampling()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.9.3 APS_wait_trigger_sampling

Waiting for sample data.

Description:

This function is used to sample data from controller. When the function is issued, the program stating to sample the information and put the data to the internal buffer. Until the trigger signal is turned on, program fetched a mass of data which size is pre-trigger length from internal buffer to the user's data buffer and continuous sample the data until reach the length that users designated. In other hand, if the timeout time is reached and the trigger signal does not raised, this function will be timeout and return an error message.

Syntax:

```
I32 APS_wait_trigger_sampling( I32 Board_ID, I32
     Length, I32 PreTrgLen, I32 TimeOutMs,
     STR_SAMP_DATA_4CH *DataArr );
```

Parameters:

`I32 Board_ID`: ID of the target controller. It's retrieved by successful call to APS_initial().

`I32 Length`: The number of sampling data. (array size)

`I32 PreTrgLen`: Pre-trigger length.

`I32 TimeOutMs`: Timeout time. Unit in millisecond.

`STR_SAMP_DATA_4CH *DataArr`: Get sampling data structure array. Array size must be larger than the parameter "Length".

Return Values:

I32 error codes: Please refer to the error code table on page 253.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
//... initial card.
APS_set_sampling_param( Board_ID, SAMP_PA_RATE, 2
     ); //Set sampling rate
APS_set_sampling_param( Board_ID, SAMP_PA_EDGE, 0
     ); //Set trigger edge (rising edge)
APS_set_sampling_param( Board_ID, SAMP_PA_LEVEL,
     1  ); //Set trigger level ( 1)
APS_set_sampling_param( Board_ID, SAMP_PA_TRIGCH,
     0  ); //Set trigger channel (channel 0)
APS_set_sampling_param( Board_ID,
     SAMP_PA_SRC_CH0, SAMP_CMD_VEL ); //Set
     channel_0 sampling source.
APS_set_sampling_param( Board_ID,
     SAMP_PA_SRC_CH1, SAMP_MIO_INP  ); //Set
     channel_1 sampling source.
I32 Length = 1024; //Total sampling data array
     size.
I32 PreTrgLen = 100; //The number of pre-trigger
     points
STR_SAMP_DATA_4CH DataArr[1024];
I32 TimeOutMs = 10000; //10 second timeout
Ret =APS_wait_trigger_sampling( Board_ID, Length,
     PreTrgLen, TimeOutMs, DataArr );
//…
```

See also:

```
APS_set_sampling_param()
APS_get_sampling_param();
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.10 Point table

6.10.1 APS_set_point_table

Set point table move parameters

Description:

This function is used to set a set of point table parameters to specified axis. Each axis has 32 set can be saved in card's memory.

Syntax:

```
I32 APS_set_point_table( I32 Axis_ID, I32 Index,
     POINT_DATA *Point );
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 Index: Specified point index to be set. Range: 0 ~ 31

POINT_DATA *Point: Structure of point table parameters. Define in "type_def.h"

```
typedef struct
{
      I32 i32_pos;//(Center)Position data (could
      be relative or absolute value) (pulse)
      I16 i16_accType;//Acceleration pattern 0: T
      curve, 1:S curve
      I16 i16_decType;// Deceleration pattern 0: T
      curve, 1:S curve
      I32 i32_acc;//Acceleration rate ( pulse /
      s2)
      I32 i32_dec;//Deceleration rate ( pulse / s2
      )
      I32 i32_initSpeed;//Start velocity( pulse /
      s )
      I32 i32_maxSpeed;//Maximum velocity  ( pulse
      / s )
      I32 i32_endSpeed; //End velocity( pulse / s
      )
      I32 i32_angle;//Arc move angle ( degree, -
      360 ~ 360 )
      U32 u32_dwell;//dwell times ( unit: ms )
      *Divided by system cycle time.
      I32 i32_opt;    //Point move option. (*)
} POINT_DATA;
```

(*) Point move option: **i32_opt**

7	6	5	4	3	2	1	Bit: 0
-	-	Last Point	Finish Condition	-	Linear/ Arc	-	Absolute/ Relative

Bit 0: 1:Relative move, 0:Absolute move

Bit 2: 1:Arc move, 0:Linear move

Bit 4: 1:INP ON(In position signal), 0:CSTP ON(command stop signal)

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Bit 5: 1: Last point index. 0: Not Last point index (if this bit is turned on, the point table move will stop after this point.)

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

```
#include "type_def.h"
#include "APS_define.h"
#include "APS168.h"
#include "ErrorCodeDef.h"

I32 ret;
POINT_DATA Point;

Point.i32_pos = 10000;//(Center)Position data
      (could be relative or absolute value)
      (pulse)
Point.i16_accType = 1;//Acceleration pattern 0: T
      curve, 1:S curve
…
//Set point data to card memory.
ret = APS_set_point_table(Axis_ID, 0, &Point );
if( ret != ERR_NoError )
{ //Error occured
}
```

See also:

```
APS_get_point_table()
APS_point_table_move()
APS_get_next_point_index()
APS_get_start_point_index()
APS_get_end_point_index()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.10.2 APS_get_point_table

Get point table move parameters

Description:

This function is used to get a set of point table parameters to specified axis. Each axis has 32 set can be saved in card's memory.

Syntax:

```
I32 APS_get_point_table( I32 Axis_ID, I32 Index,
     POINT_DATA *Point );
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 Index: Specified point index to be set. Range: 0 ~ 31

POINT_DATA *Point: Structure of point table parameters. Define in “type_def.h”

```
typedef struct
{
      I32 i32_pos;//(Center)Position data (could
      be relative or absolute value) (pulse)
      I16 i16_accType;//Acceleration pattern 0: T
      curve, 1:S curve
      I16 i16_decType;// Deceleration pattern 0: T
      curve, 1:S curve
      I32 i32_acc;//Acceleration rate ( pulse /
      s2)
      I32 i32_dec;//Deceleration rate ( pulse / s2
      )
      I32 i32_initSpeed;//Start velocity( pulse /
      s )
      I32 i32_maxSpeed;//Maximum velocity  ( pulse
      / s )
      I32 i32_endSpeed; //End velocity( pulse / s
      )
      I32 i32_angle;//Arc move angle ( degree, -
      360 ~ 360 )
      U32 u32_dwell;//dwell times ( unit: ms )
      *Divided by system cycle time.
      I32 i32_opt;     //Point move option. (*)
} POINT_DATA;
```

(*) Point move option: i32_opt

7	6	5	4	3	2	1	Bit : 0
-	-	Last Point	Finish Condition	-	Linear/ Arc	-	Absolute/ Relative

Bit 0: 1:Relative move, 0:Absolute move

Bit 2: 1:Arc move, 0:Linear move

Bit 4: 1:INP ON(In position signal), 0:CSTP ON(command stop signal)

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Bit 5: 1: Last point index. 0: Not Last point index (if this bit is turned on, the point table move will stop after this point.)

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

```
#include "type_def.h"
#include "APS_define.h"
#include "APS168.h"
#include "ErrorCodeDef.h"

//... initial card.
I32 ret ;
POINT_DATA Point;
ret =APS_get_point_table( Axis_ID, 0, &Point );
if( ret != ERR_NoError )
{
     //Error.
}
```

See also:

```
APS_set_point_table()
APS_point_table_move()
APS_get_next_point_index()
APS_get_start_point_index()
APS_get_end_point_index()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.10.3 APS_point_table_move

Start a point table move

Description:

This function is used to start a point table move. When point table move is started, the system will take the point parameters one by one from "**StartIndex**" to "**EndIndex**". Therefore user must specified the point parameters to point table before perform point table move.

When the axis is in point table moving, user cannot perform others move until point table move is finish.

User could uses stop_move, emg_stop, function to forced stop point table move.

Relative motion status description		
PMV	Point table move state	(Control axis) ON: in point table move state
PDW	Point table Dwell state	(Control axis) ON: in point table dwell state
PPS	Point table pause state	(Control axis) ON: in point table pause state
SLV	Slave axis move state	(Slave axis) ON: in slave axis move state

Reference axis: The first axis in axis array. User can specify it.

Control axis: The minimum axis ID will be the control axis.

Slave axis: Other axes except control axis.

For example:

Ex1.

```
I32 AxisArray[4] = {3, 1, 2, 4 };
Control axis is ID= 1.
Reference axis is ID = 3.
Slave axes are ID = 2, 3, 4
```

Ex2.

```
I32 AxisArray[3] = { 1, 2, 4 };
Control axis is ID= 1.
Reference axis is ID = 1.
Slave axes are ID = 2, 4
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Syntax:

```
I32 APS_point_table_move( I32 Dimension, I32
    *Axis_ID_Array, I32 StartIndex, I32 EndIndex
    );
```

Parameters:

`I32 Dimension`: Dimension of axis array. (Linear move :1 ~ 4), (Arc move: 2)

`I32 *Axis_ID_Array`: Axis ID array.

`I32 StartIndex`: The first running point index.

`I32 EndIndex`: The end of point index. (Last point index + 1).

<Ex>

StartIndex = 3, EndIndex = 6.

The running sequence will be 3 -> 4 -> 5

Return Values:

I32 error codes: Please refer to the error code table on page 253.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
#include "type_def.h"
#include "APS_define.h"
#include "APS168.h"
#include "ErrorCodeDef.h"

I32 ret;
POINT_DATA Point;
I32 Axis_ID_Array;

Point.i32_pos = 10000;//(Center)Position data
      (could be relative or absolute value)
      (pulse)
Point.i16_accType = 1;//Acceleration pattern 0: T
      curve, 1:S curve
…
//Set point data to card memory.
ret = APS_set_point_table(Axis_ID, 0, &Point );
…
if( ret != ERR_NoError )
{ //Error occured
}

// Start a point table move.
Axis_ID_Array = Axis_ID;
ret = APS_point_table_move( 1, &Axis_ID_Array, 0
      , 3 );
…
```

See also:

```
APS_set_point_table()
APS_get_point_table()
APS_point_table_move()
APS_get_next_point_index()
APS_get_start_point_index()
APS_get_end_point_index()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.10.4 APS_get_running_point_index

Get next point move index when axis is perform a point move

Description:

This function is used to get the running point index when the axis is performing a point table move. For example, if the system is running index 3, this function will return index = 3.

If the operation is running at the last point, this function will return the “end point index”.

Note: At the system state start, the default value is -1

Syntax:

```
I32 APS_get_running_point_index( I32 Axis_ID, I32
     *Index );
APS_get_running_point_index( ByVal Axis_ID As
     Long, Index As Long) As Long
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535.

`I32 *Index`: return next running point index.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

```
#include “type_def.h”
#include “APS_define.h”
#include “APS168.h”
#include “ErrorCodeDef.h”
//... initial card.
//…start network

I32 Index;
I32 ret = APS_get_running_point_index ( Axis_ID,
     &Index );
If( ret != ERR_NoError )
{ //Error occured
}
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

See also:

```
APS_set_point_table()
APS_get_point_table()
APS_point_table_move()
APS_get_start_point_index()
APS_get_end_point_index()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.10.5 APS_get_start_point_index

Get the first point move index when axis is perform a point move

Description:

This function is used to get the first point index when the axis is performing a point table move.

Syntax:

```
I32 APS_get_start_point_index( I32 Axis_ID, I32
     *Index );
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535.

`I32 *Index`: return the first running point index.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

```
#include "type_def.h"
#include "APS_define.h"
#include "APS168.h"
#include "ErrorCodeDef.h"
//... initial card.
//…start network

I32 Index;
I32 ret = APS_get_start_point_index ( Axis_ID,
     &Index );
If( ret != ERR_NoError )
{ //Error occured
}
```

See also:

```
APS_set_point_table();
APS_get_point_table();
APS_point_table_move();
APS_get_next_point_index();
APS_get_end_point_index();
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.10.6 APS_get_end_point_index

Get the end of point move index when axis is perform a point move

Description:

This function is used to get the end of point index when the axis is performing a point table move.

Syntax:

```
I32 APS_get_end_point_index( I32 Axis_ID, I32
     *Index );
```

Parameters:

I32 Axis_ID: The Axis ID from 0 to 65535.

I32 *Index: return the end of running point index.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

```
#include "type_def.h"
#include "APS_define.h"
#include "APS168.h"
#include "ErrorCodeDef.h"

//... initial card.
//…start network

I32 Index;
I32 ret = APS_get_end_point_index( Axis_ID,
     &Index );
If( ret != ERR_NoError )
{ //Error occured
}
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

See also:

```
APS_set_point_table();
APS_get_point_table();
APS_point_table_move();
APS_get_next_point_index();
APS_get_start_point_index();
```

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.10.7 APS_set_table_move_pause

Pause point table move

Description:

This function is used to pause the point table move. When pause command is issued, it will not stop current point but stop at next point index.

Syntax:

```
I32 APS_set_table_move_pause( I32 Axis_ID, I32
     Pause_en );
APS_set_table_move_pause(ByVal Axis_ID As Long,
     ByVal Pause_en As Long ) As Long
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535.

`I32 Pause_en`:

- 1: Pause
- 0: Not paused.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
#include "type_def.h"
#include "APS_define.h"
#include "APS168.h"
#include "ErrorCodeDef.h"

//... initial card.
//…start network

I32 Index;
I32 ret = APS_set_table_move_pause ( Axis_ID, 1
     ); //Pause point table move
If( ret != ERR_NoError )
{ //Error occured
}
```

See also:

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.10.8 APS_set_table_move_repeat

Set point table move repeat

Description:

This function is used to set point table move repeat. When repeat function is enabled, it will repeat the point move until repeat function is disabled or stop function is issued.

Syntax:

```
I32 APS_set_table_move_repeat ( I32 Axis_ID, I32
     Repeat_en );
APS_set_table_move_pause(ByVal Axis_ID As Long,
     ByVal Repeat_en As Long ) As Long
```

Parameters:

`I32 Axis_ID`: The Axis ID from 0 to 65535.

`I32 Repeat_en`:

- 1: Repeat.
- 0: Don't repeat.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
#include "type_def.h"
#include "APS_define.h"
#include "APS168.h"
#include "ErrorCodeDef.h"

//... initial card.
//…start network

I32 Index;
I32 ret = APS_set_table_move_pause ( Axis_ID, 1
     ); //Pause point table move
If( ret != ERR_NoError )
{ //Error occured
}
```

See also:

6.11 Field bus

6.11.1 APS_set_field_bus_param

Set field bus related parameters

Description:

This function is used to set field bus parameters for initial communication. Users must set this function before starting field bus communication. Otherwise, the field bus will be started by default.

Syntax:

```
I32 APS_set_field_bus_param( I32 Board_ID, I32
     BUS_No, I32 BUS_Param_No, I32  BUS_Param );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 BUS_No: Field bus number.(Port number) value: 0~1

I32 BUS_Param_No: Field bus parameter number, Refer to table defination

I32 BUS_Param: Field bus parameter data. Refer to table definition.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

See also:

```
APS_get_field_bus_param()
APS_start_field_bus()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.11.2 APS_get_field_bus_param

Get field bus related parameters

Description:

This function is used to get field bus parameters for initial communication.

Syntax:

```
I32 APS_get_field_bus_param( I32 Board_ID, I32
     BUS_No, I32 BUS_Param_No, I32 *BUS_Param );
```

Parameters:

`I32 Board_ID`: ID of the target controller. It's retrieved by successful call to APS_initial().

`I32 BUS_No`: Field bus number.(Port number) value: 0~1

`I32 BUS_Param_No`: Field bus parameter number, Refer to table defination

`I32 *BUS_Param`: Return field bus parameter data. Refer to table definition.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

See also:

`APS_set_field_bus_param`

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.11.3 APS_start_field_bus

Start the network of specified field bus

Description:

This function is used to start field bus communication. Once it is started, it will search all modules connected to the port. There could be motion axis on the port, so the starting axis ID must be passed into the function when using this function. All axes of the port will get the axis ID offset from the starting axis ID.

You should call this function before other field operating function such as digital I/O functions.

APS_stop_field_bus() function must be called at the end of filed bus operation.

Syntax:

```
I32 APS_start_field_bus( I32 Board_ID, I32
     BUS_No, I32 Starting_Axis_ID );
APS_start_field_bus( ByVal Board_ID As Long,
     ByVal BUS_No As Long, ByVal Starting_Axis_ID
     As Long ) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 BUS_No: Field bus number.(Port number) value: 0~1

I32 Starting_Axis_ID: Starting axis ID number of this field bus number. (Reserved)

Return Values:

I32 error codes: Please refer to the error code table on page 253.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
I32 ret; //Return error code.
I32 boardId = 0;
I32 busNum = 0; //Bus number.
I32 startingAxisId = 1000; //Startin axis ID of
     the filed bus.

ret = APS_start_field_bus( boardId, busNum,
     startingAxisId );
// Field bus operation…
APS_stop_field_bus(boardId, busNum ); //Stop
     field bus.
```

See also:

`APS_stop_field_bus()`

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.11.4 APS_stop_field_bus

Stop the network of specified field bus

Description:

This function is used to stop field bus communication and release its resource.

This function must be called at end of process, if user used **APS_start_field_bus**() to start network.

Syntax:

```
I32 APS_stop_field_bus( I32 Board_ID, I32 BUS_No
    );
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 BUS_No: Field bus number.(Port number) value: 0~1

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

```
I32 ret; //Return error code.
I32 boardId = 0;
I32 busNum = 0; //Bus number.
I32 startingAxisId = 1000; //Startin axis ID of
    the filed bus.

ret = APS_start_field_bus( boardId, busNum,
    startingAxisId );
// Field bus operation…
APS_stop_field_bus(boardId, busNum ); //Stop
    field bus.
```

See also:

```
APS_start_field_bus()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.11.5 APS_field_bus_d_set_output

Set field bus digital output

Description:

This function is used to set field bus digital output modules. The maximum data length of one module ID is 32-bit. If the module ID has fewer channels than 32, the higher bit must be remained zero when outputting. The read back data of higher bit will be zero.

Syntax:

```
I32 APS_field_bus_d_set_output( I32 Board_ID, I32
     BUS_No, I32 MOD_No, I32 DO_Value );
APS_field_bus_d_set_output( ByVal Board_ID As
     Long, ByVal BUS_No As Long, ByVal MOD_No As
     Long, ByVal DO_Value As Long )As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 BUS_No: Field bus number.(Port number) value: 0~1

I32 MOD_No: Module number.

For High Speed Link(HSL) type field bus, the range of module number is 0 to 63.

I32 DO_Value: Digital output value. In bit format. Bit 0 corresponding to digital output channel 0 and the rest may be deduced by analogy.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

Example:

```
I32 ret; //return error code.
I32 boardId = 0;
I32 busNum = 0;
I32 moduleNum = 0;
I32 DO_Value = 0;
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

```
//Start Field bus first.
//  ret = APS_start_field_bus( boardId, busNum,
     startingAxisId );
DO_Value = 0xF;
ret = APS_field_bus_d_set_output(boardId,
     busNum,, moduleNum, DO_Value );
```

See also:

`APS_field_bus_d_get_output()`

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.11.6 APS_field_bus_d_get_output

Get field bus digital output

Description:

This function is used to get field bus digital output modules. Some module ID can't read back the output information. Please check each module's hardware specification. The maximum data length of one module ID is 32-bit. If the module ID has fewer channels than 32, the higher bit must be remained zero when outputting. The read back data of higher bit will be zero.

Syntax:

```
I32 APS_field_bus_d_get_output( I32 Board_ID, I32
     BUS_No, I32 MOD_No, I32 *DO_Value );
APS_field_bus_d_get_output( ByVal Board_ID As
     Long, ByVal BUS_No As Long, ByVal MOD_No As
     Long, DO_Value As Long ) As Long
```

Parameters:

I32 Board_ID: ID of the target controller. It's retrieved by successful call to APS_initial().

I32 BUS_No: Field bus number.(Port number) value: 0~1

I32 MOD_No: Module number.

For High Speed Link(HSL) type field bus, the range of module number is 0 to 63.

I32 *DO_Value: Return digital output value. Bit 0 corresponding to digital output channel 0 and the rest may be deduced by analogy.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
I32 ret; //return error code.
I32 boardId = 0;
I32 busNum = 0;
I32 moduleNum = 0;
I32 DO_Value = 0;

//Start Field bus first.
//  ret = APS_start_field_bus( boardId, busNum,
      startingAxisId );

ret = APS_field_bus_d_get_output(boardId, busNum,
      moduleNum, &DO_Value );
```

See also:

```
APS_field_bus_d_set_output();
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.11.7 APS_field_bus_d_get_input

Get field bus digital input

Description:

This function is used to get input data from field bus digital input modules. The maximum data length of one module ID is 32-bit. If the module ID has fewer channels than 32, the higher bit must be remained zero.

Syntax:

```
I32 APS_field_bus_d_get_input( I32 Board_ID, I32
     BUS_No, I32 MOD_No, I32 *DI_Value );
APS_field_bus_d_get_input( ByVal Board_ID As
     Long, ByVal BUS_No As Long, ByVal MOD_No As
     Long, DI_Value As Long ) As Long
```

Parameters:

`I32 Board_ID`: ID of the target controller. It's retrieved by successful call to APS_initial().

`I32 BUS_No`: Field bus number.(Port number) value: 0~1

`I32 MOD_No`: Module number.

`I32 *DI_Value`: Return digital input value.

Return Values:

I32 error codes: Please refer to the error code table on page 253.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Example:

```
I32 ret; //return error code.
I32 boardId = 0;
I32 busNum = 0;
I32 moduleNum = 0;
I32 DI_Value = 0;

//Start Field bus first.
//ret = APS_start_field_bus( boardId, busNum,
      startingAxisId );
ret = APS_field_bus_d_get_input(boardId, busNum,,
      moduleNum, &DI_Value );
```

See also:

```
APS_field_bus_d_set_output()
APS_field_bus_d_get_output()
```

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.12 Definition Tables

We provide C/C++ standard header file which define all of SSC-NET parameters number. This is a safe and clear way of coding your application. The actual numeric value of these constants may change in future versions of the API, however the definition of symbols will remain fixed. Use of these symbolic values will help to insure that future changes to the API won't break existing code.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.12.1 Board Parameters Definition Table

Value	Definination	Description	Parameter data meaning	Note
0000 0000h	PRB_EMG_LOGIC	EMG logic setting	0: Normal close (default) 1: Normal open	
SSCNET parameter				
Value	Definination	Description	Parameter data meaning	Note
0001 0000h	PRB_SSC_CYCLE_TIME	SSCNET 3 communi-cation cycle time setting	0: 0.888ms (default) 1: 0.444ms	a

a. This parameter is valid after restart SSCNET network.

Table 6-1: Board Parameters Definition Table

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.12.2 Axis Parameter Definition Table

NO.	Define	Description:	Value	Notes
00h	PRA_EL_LOGIC	PEL/MEL input logic	0:Not inverse (default) 1:Inverse	
01h	PRA_ORG_LOGIC	ORG input logic	0:Not inverse (default) 1:Inverse	
02h	PRA_EL_MODE	The mode of stop when end limit ON	0: Deceleration stop 1: Stop immediately	
03h	PRA_MDN_CONDI	Motion done condition (Affective with motion stats MOT bit)	0: Control command done (default) 1: Command done with INP 2: Command done with ZSP 3: Command done with INP & ZSP	
04h	Reserved			
05h	Reserved			
06h	Reserved			
10h	PRA_HOME_MODE	Home mode setting	0: home mode 1 (default)	
11h	PRA_HOME_DIR	Homing direction	0: positive direction (default) 1: negative direction	
12h	PRA_HOME_CURVE	Home move acceleration / Deceleration speed pattern	0: T-curve (default) 1: S-curve	
13h	PRA_HOME_ACC	Home move acceleration/Deceleration rate	Unit: pulse/sec2	
14h	PRA_HOME_VS	Homing start velocity	Unit: pulse/sec	
15h	PRA_HOME_VM	Homing maximum velocity	Unit: pulse/sec	
16h	Reserved			a
17h	Reserved			
18h	PRA_HOME_EZC	Enable EZ signal alignment	0: Disable (Default) 1: Enable	
19h	PRA_HOME_VO	Homing leave home velocity	Unit: pulse/sec	
1Ah-1Fh	Reserved			
20h	PRA_CURVE	Acceleration / Deceleration speed pattern	0: T-Curve (Default) 1: S-Curve	

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

21h	PRA_ACC	Acceleration rate	Unit: pulse/sec2	
22h	PRA_DEC	Deceleration rate	Unit: pulse/sec2	
23h	PRA_VS	Start velocity	Unit: pulse/sec	
24h	Reserved			
25h	PRA_VE	End velocity	Unit: pulse/sec	
26h~2Fh	Reserved			
30h	PRA_PT_PAUSE	Paused point table move	1: Paused 0: Not paused (Default)	b
31h	PRA_PT_REPEAT	Repeat point table move	1: Repeat 0: Not repeat (Default)	
32h~3Fh	Reserved			
40h	PRA_JG_MODE	Jog mode	0: Free mode(Default) 1: step mode	
41h	PRA_JG_DIR	Jog move direction	0: Positive direction (Default) 1:negative direction	
42h	PRA_JG_CURVE	Jog speed pattern	0: T-curve 1:S-curve	
43h	PRA_JG_ACC	Acceleration rate	Unit: pulse/sec2	
44h	PRA_JG_DEC	Deceleration rate	Unit: pulse/sec2	
45h	PRA_JG_VM	Max. velocity	Unit: pulse/sec	
46h	PRA_JG_OFFSET	Step offset	Unit: pulse (For step mode)	
47h	PRA_JG_DELAY	Delay time	Unit: ms (cycle time alignment) (For step mode)	

a. Do not set any parameter data.
b. Reset to default value when start network.

Table 6-2: Axis Parameter Definition Table

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.12.3 SSCNET Axis Parameters Definition Table

NO.	Define	Description:	Value	Note
0001 0000h	PRA_SSC_SERVO_PARAM_SRC	Select servo parameter source when start SSC-NET	0: Do not update(Default) 1: Default value. 2: Flash memory	
0001 0001h	PRA_SSC_SERVO_ABS_POS_OPT	Enable absolute position system.	0: Disable. 1: Enable absolute position system	
0001 0002h	PRA_SSC_SERVO_ABS_CYC_CNT	Absolute cycle counter of servo driver	0 ~ 65535 (16 bit)	
0001 0003h	PRA_SSC_SERVO_ABS_RES_CNT	Absolute resolution counter of servo driver	0~262143 (18bit)	
0001 0004h	PRA_SSC_TORQUE_LIMIT_P	Positive torque limit value (0.1%)	0~32767 (default 3000)	a
0001 0005h	PRA_SSC_TORQUE_LIMIT_N	Negative torque limit value (0.1%)	0~32767 (default 3000)	
0001 0006h	PRA_SSC_TORQUE_CTRL	Torque control enable	0: Disable, (Default, Control with motor max. torque) 1: Enable, (Control with torque limit value)	b
0001 0007h	PRA_SSC_RESOLUTION	E-gear factor2Value = resolution	Value = 12-18 (Default 18, resolution = 262144)	c

a. 0.1% Set 1000 mean 100%
b. Some SSCNET axis parameters will be rest to default value when you start SSCNET network.
c. This parameter is valid after re-start SSCNET network.

Table 6-3: SSCNET Axis Parameters Definition Table

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.12.4 Sampling Parameters Definition Table

Para NO.	Definition	Description:	Parameter data value.
00h	SAMP_PA_RATE	Sampling rate(cycle) (depended on cycle time)	1- 65535(times of cycle)
02h	SAMP_PA_EDGE	Edge triggered	0:Rising edge, 1:faling edge
03h	SAMP_PA_LEVEL	Triggered level	(I32) -2147483648 to 2147483647
05h	SAMP_PA_TRIGCH	Trigger channel	0 - 3 (Ch0-Ch3)
10h	SAMP_PA_SRC_CH0	Sampling source of Channel 0	Refer to sampling source table
11h	SAMP_PA_SRC_CH1	Sampling source of Channel 1	Refer to sampling source table
12h	SAMP_PA_SRC_CH2	Sampling source of Channel 2	Refer to sampling source table
13h	SAMP_PA_SRC_CH3	Sampling source of Channel 3	Refer to sampling source table

Table 6-4: Sampling Parameters Definition Table

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.12.5 Sampling Source Definition Table

Source	Symbol Define	Description:	Value range
00h	SAMP_COM_POS	Command position (pulse)	I32 value
01h	SAMP_FBK_POS	Feedback position (pulse)	I32 value
02h	SAMP_CMD_VEL	Command velocity (pps)	I32 value
03h	SAMP_FBK_VEL	Feedback velocity (pps)	I32 value
04h	SAMP_MIO	motion IO status (Same as Get motion IO function)	I32 value (bit format)
05h	SAMP_MSTS	Motion status (Same as Get motion status function)	I32 value (bit format)
06h	SAMP_MSTS_ACC	Motion status at acceleration (Command velocity)	0: Not at acceleration 1: At acceleration
07h	SAMP_MSTS_MV	Motion status at max velocity (Command velocity)	0: Not at max. velocity 1: At max. velocity
08h	SAMP_MSTS_DEC	Motion status at deceleration (Command velocity)	0: Not at deceleration 1: At deceleration
09h	SAMP_MSTS_CSTP	Motion status command stop (CSTP)	0: CSTP status ON 1: CSTP status OFF
0Ah	SAMP_MSTS_NSTP	Motion status normal stop (NSTP)	0: NSTP status ON 1: NSTP status OFF
0Bh	SAMP_MIO_INP	Motion status in position (INP)	0: INP status ON 1: INP status OFF
0Ch	SAMP_MIO_ZERO	Motion status zero (ZERO)	0: ZERO status ON 1: ZERO status OFF
0Dh	SAMP_MIO_ORG	Motion status ORG status	0: OGR status ON 1: OGR status OFF

Table 6-5: Sampling Source Definition Table

Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.12.6 Motion IO Status Bit Definition

7	6	5	4	3	2	1	0
SVON	INP	EZ	EMG	ORG	MEL	PEL	ALM
15	14	13	12	11	10	9	8
	ABSL	TLC	SMEL	SPEL	ZERO	WARN	RDY

Bit number description

Bit	Define	Description:
0	ALM	Servo alarm
1	PEL	Positive end limit
2	MEL	Negative end limit
3	ORG	Original position sensor(home sensor)
4	EMG	EMG sensor
5	EZ	EZ passed
6	INP	In position
7	SVON	Servo ON
8	RDY	Ready
9	WARN	Warning
10	ZERO	Zero speed, The zero speed output range setting, please refer to the manual of servo driver. (Servo parameter PC07)
11	SPEL	Software positive end limit
12	SMEL	Software negative end limit
13	TLC	Torque is limited by torque limit value. (When torque control is turned ON)
14	ABSL	Absolute position lost.

Table 6-6: Motion IO Status Bit Definition

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.12.7 Motion Status Definition Table

7	6	5	4	3	2	1	0
SMV	HMV	NSTP	DIR	DEC	ACC	VM	CSTP
15	14	13	12	11	10	9	8
JOG	SLV	PPS	PDW	PMV	SMO	CIP	LIP
23	22	21	20	19	18	17	16
SEMGS	MELS	PELS	WANS	ALMS	EMGS	SVONS	ASTP
31	30	29	28	27	26	25	24
	ERRS				STPOA	SMELS	SPELS

Bit number description

Bit	Define	Description
0	CSTP	Command stopped
1	VM	At maximum velocity
2	ACC:	At acceleration
3	DEC:	At deceleration
4	DIR:	Move direction. 1:Positive direction, 0:Negative direction
5	NSTP	Normal stop(Motion done)
6	HM	In homing
7	SMV	Single axis move(relative, absolute, velocity move)
8	LIP	Linear interpolation
9	CIP	Circular interpolation
10	SMO	Single move
11	PMV	Point table move
12	PDW	Point table dwell move
13	PPS	Point table pause state
14	SLV	Slave axis move
15	JOG	Jog move
16	ASTP	Abnormal stop
17	SVONS	Servo off stopped
18	EMGS	EMG stopped
19	ALMS	Alarm stop
20	WANS	Warn stopped

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

21	PELS	PEL stopped
22	MELS	MEL stopped
23	SEMGS	software EMG stopped
24	SPELS	software PEL stopped
25	SMELS	software MEL stopped
26	STPOA	Stop by others axes
27	--	Reserved
28	--	Reserved
29	--	Reserved
30	ERRS	Error command stopped

Table 6-7: Motion Status Definition Table

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.12.8 Interrupt Factor Table

Item	Description:
0	Axis 0 interrupt factors
1	Axis 1 interrupt factors
…	…
15	Axis 15 interrupt factors
16	System interrupt factors

Item 0~15: Axes interrupt factors number definition

7	6	5	4	3	2	1	0
IZERO	IWARN	IINP	IEZ	IORG	IMEL	IPEL	IALM
15	14	13	12	11	10	9	8
--	ITLC	IASTP	INSTP	IDEC	IACC	IVM	ICSTP
23	22	21	20	19	18	17	16
--	--	--	--	--	--	--	--
31	30	29	28	27	26	25	24
--	--	--	--	--	--	--	--

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Axes interrupt factors condition description:

NO.	Define	Interrupt condition description
0	IALM	Servo alarm signal turn ON
1	IPEL	Positive end limit switch is turn ON
2	IMEL	Minus (Negative) end limit switch turn ON
3	IORG	Home switch turn ON
4	IEZ / IEZP	EZ passed signal turn ON
5	IINP	In position signal turn ON
6	IWARN	Servo warning ON
7	IZERO	Zero speed
8	ICSTP	Command stop
9	IVM	In maximum velocity
10	IACC	In acceleration
11	IDEC	In deceleration
12	INSTP	Normal stop(Motion done)
13	IASTP	Abnormal stop
14	ITLC	Torque limit control is turn ON
15	Reserved	--

Table 6-8: Interrupt Factor Table

Note: (1) In SSCNET system, When zero position signal(EZ) from servo driver is ON, EZP bit will ON even if EZ is turn OFF.

(2), INSTP: Axis is stopped normally. If axis is stopped abnormally such as emergency stop and Limit switch on stop etc, this interrupt factor will not be triggered. All motion action including home move which can be waited motion done by this interrupt factor.

You can set normal stop (motion done) condition by set axis parameter function.

CSTP: Motion command is stopped, but the axis could be still in motion.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.12.9 Function Return Code

The following table provides a list of possible return value in our library. If the return value is a negative value, it means there are some errors or warning occurred.

We provide C/C++ standard header file, "**`ErrorCodeDef.h`**", which define all errors return value.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

6.13 Error Code Table

Code	Definition	Error descriptions and items to check
0	ERR_NoError	Success, No error
-1	ERR_OSVersion	Operating system version error. The current operating system you used are not supported by this function.
-2	ERR_OpenDriverFailed	Open driver failed. Create driver interface failed. Check device driver is installed correctly. Check devices are installed correctly in your system.
-3	ERR_InsufficientMemory	System memory insufficiently. There is not enough memory in your system.
-4	ERR_DeviceNotInitial	The Device or the card is not be initialized. Check the card IDThe device has been closed. The device is not be initialized.
-5	ERR_NoDeviceFound	Devices not found. Check device driver is installed correctly. Check devices are installed correctly in your system.
-6	ERR_CardIdDuplicate	Card ID duplicated. Check the card ID settings (SW jump). Check the parameter of initial function is correctly.
-7	ERR_DeviceAlreadyInitaled	The devices have already been initialed. 1. Check the close card function is work correctly.
-8	ERR_InterruptNotEnable	Interrupt events not be enabled. 1. Enable the hardware interrupt. 2. Check the interrupt factor is set correctly.
-9	ERR_TimeOut	Function timeout.
-10	ERR_ParametersInvaild	The value of the parameters is incorrect. Check the setting range of parameters. Compare the setting value of parameters with user manual.
-11	ERR_SetEEPROM	Hardware memory write error.
-12	ERR_GetEEPROM	Hardware memory read error.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000

Code	Definition	Error descriptions and items to check
-13	ERR_FunctionNotAvailable	The function is not available in current stage. The device is not support this function. System is in error state. 1. Check the function library. 2. Check the hardware connection (servo drive connection) 3. Reinitialize (Reboot) the system.
-14	ERR_FirmwareError	Firmware process error. 1. Check the firmware version.
-15	ERR_CommandInProcess	The previous command is in process.
-16	ERR_AxisIdDuplicate	Axes' ID is duplicated.
-1000	ERR_Win32Error	No such event number, or WIN32_API error, contact with ADLINK's FAE staff.

LRI Distribuída no Brasil pela LRI | vendas@lri.com.br | 51 2129-3000