ADLINK TECHNOLOGY INC.

PCI-8253/56
DSP-based
3/6-Axis Analog
Motion Control Card
User's Manual

Manual Rev.	2.02
Revision Date:	January 5, 2009
Part No:	50-15057-1010

Recycled Paper

Advance Technologies; Automate the World.

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):

Table of Contents

List of Tables

List of Figures

1 Introduction

The PCI-8253/6 is a DSP-based 3/6-axis analog type motion control card with PCI interface. This motion control card leverages ADLINK softmotion that offers comprehensive motion trajectory functions.

ADLINK softmotion enables simplified utilization of complex motion manipulation which involves jogging, point-to-point positioning, multiple axes synchronized motion, contouring, etc. ADLINK offers motion kernel customization service to met your specially requirement.

Comprehensive Features – The PCI-8253/6 offers many features including faster encoder speeds, high speed position comparison and trigger pulse outputs, adjustable PID plus feed-forward gain and axis parameter on-the-fly, non-volatile program and data memory, and vast dedicated onboard I/O. Through use of 32-bit high-speed dual port RAM, the PCI-8253/6 allows for high-speed servo control up to 20 million encoder counts per second. The servo also provides update rates as low as 50 μs per axis.

This motion controller also provides the programmable acceleration and deceleration to eliminate jerk and smooth velocity profile. For each axes, individual unlimited point tables with 32 depth buffers can realize seamless continuous movements. These tables are also able to combine linear and arc segment.

Closed-loop Control with PID *plus* Feed Forward Gain – The PCI-8253/6 uses closed-loop control architecture to ensure precise positioning and to control the position at anytime to minimize position errors. PID plus feed-forward gain tuning in a closed-loop system is very important on performance. The feed forward feature is a predictive type of control. To drive the error to zero fast enough to maintain precise motion and make the computing and react quickly, the feed forward control loop is pulsed with closed-loop PID. In a typical feedback control system with an inner velocity and an outer position-control loop, velocity, and acceleration, feed forward is added after the position loop to the velocity loop. Predictive control depends on earlier conditions, thus the signal has to bypass the position loop. In other words, the setpoint is compared to the next step in the process rather than the output.

Figure 1-1 shows the functional block diagram of the PCI-8253/6 card. It contains a DSP as a main chip. Around the DSP is motion interface, a FPGA and some necessary devices for DSP such as SDRAM and DPRAM.

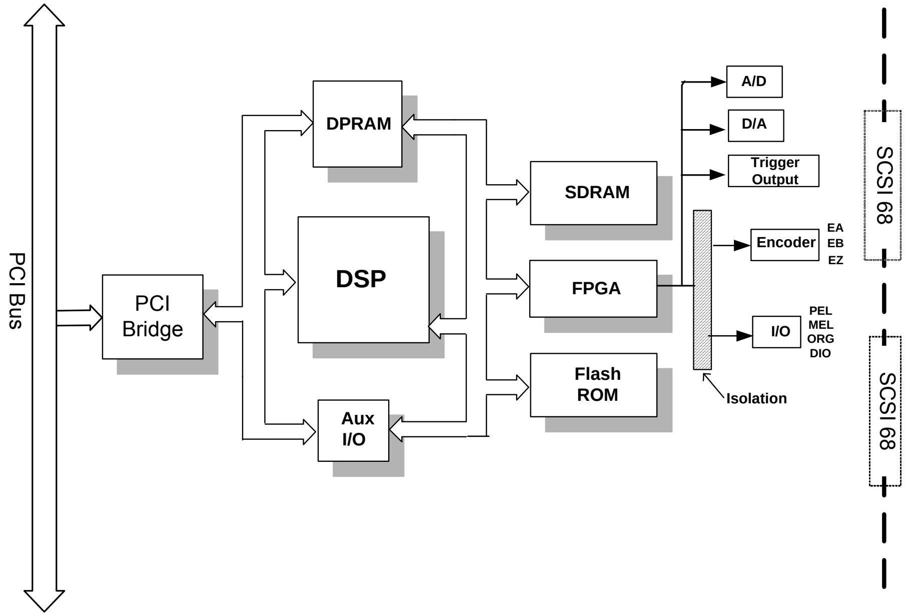
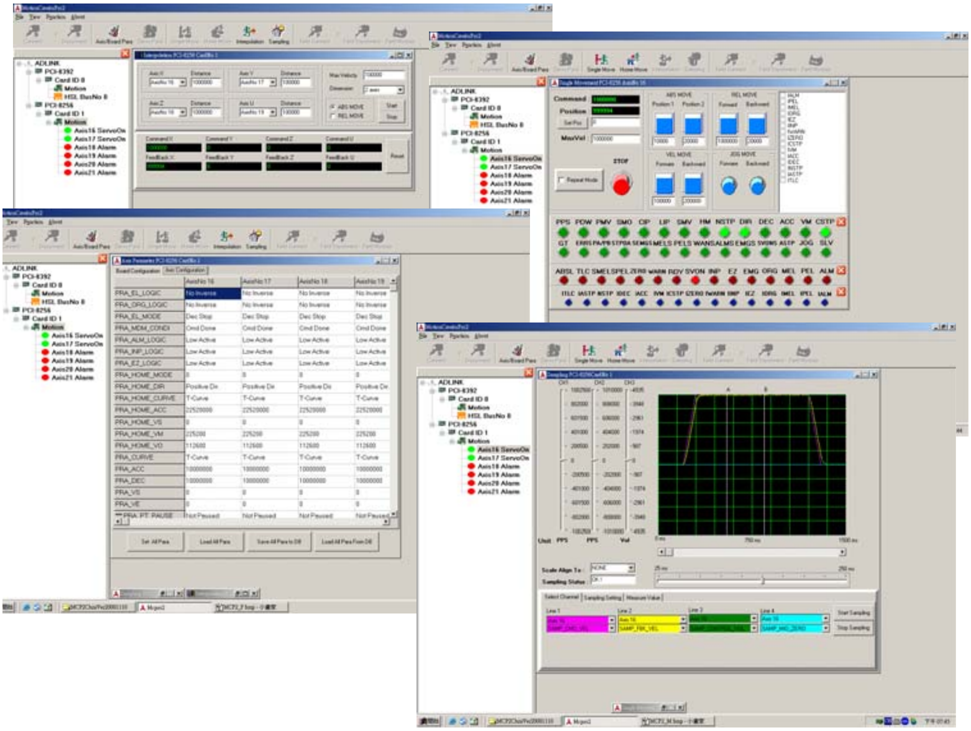


Figure 1-1: Block Diagram of the PCI-8253/6

MotionCreatorPro 2 is a Windows-based application development software package included with the PCI-8253/6. 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-8253/6. By using this utility, you can easy tune the axis parameter servo gain (PID plus feed forward gain) reducing the effort required for gain tuning. In addition, the sampling windows display more accurate motion data analysis while integrating the axis parameters and PID gain on-the-fly changes. This utility thus enables the PCI-8253/6 to provide precise positioning control with less effort.



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.

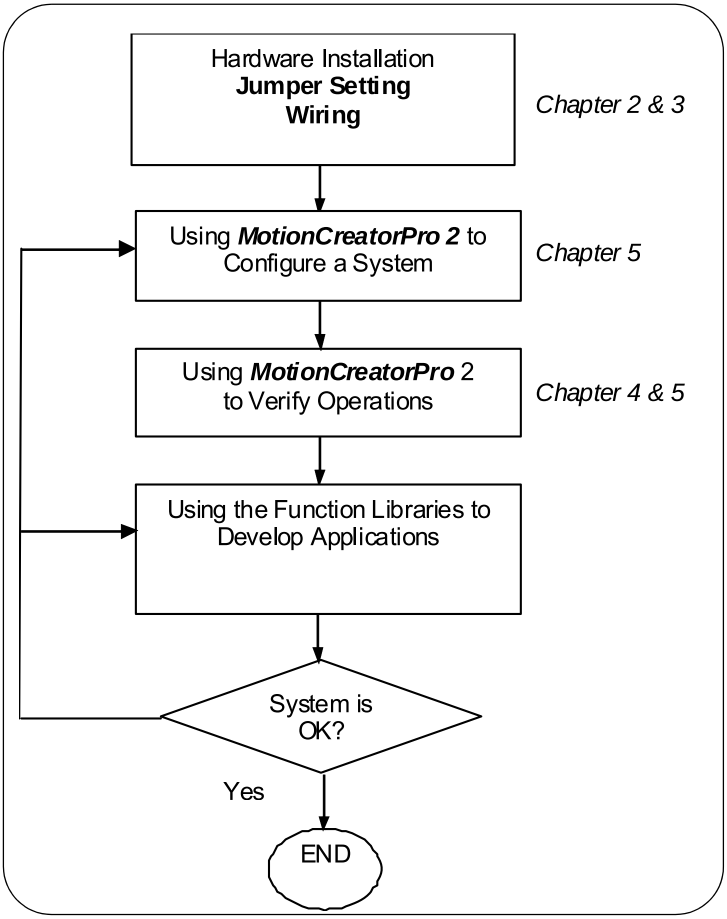

Hardware Installation
Jumper Setting
Wiring
Chapter 2 & 3
Using MotionCreatorPro 2 to Configure a System
Chapter 5
Using MotionCreatorPro 2 to Verify Operations
Chapter 4 & 5
Using the Function Libraries to Develop Applications
System is OK?
Yes
END

Figure 1-2: Flowchart for Developing an Application

1.1 Features

The following list summarizes the main features of the PCI-8253/6 motion control system.

- 32-bit PCI bus, Rev. 2.2, 33 MHz
- On-board 250 MHz DSP
- 3/6 axes of ±10 volts analog command for controlling servo motors by differential command signals
- Maximum servo update rate is less than 300 μs for 6 axes
- Built-in PID with feed-forward gain closed loop control algorithm - reducing the following error and make fast response time
- Available with 3/6 encoders – providing up to 6 axes of closed-loop control, support EA/EB and index interface
- Encoder feedback frequency up to 20 MHz
- Digital filter for encoder input to reduce noise disturbance
- 1/2 channel up to 1 MHz high speed trigger pulse output for PCI-8253/PCI-8256
- Programmable trigger pulse width from 0.3 us to 300 ms
- A/D inputs (3/6 channel, 14 bit, ±10V)
- Manual pulse generator interface; 1x, 10x, 100x, 1000x amplification
- 4-bit board ID for multiple board indexing
- One dedicated emergency input pin
- On-board 512kb flash ROM for motion kernel and non-volatile data – PID parameters
- ADLINK softmotion provides comprehensive trajectory control functions and application functions
- Programmable interrupt source control to host PC
- Dedicated motion I/O for per axis
- High speed position latch function via ORG and Index signals
- General purpose I/O: 4DI/4DO for PCI-8253 and 8DI/8DO for PCI-8256
- Watch dog timer for safety control

- Support for up to 16 cards in one system
- Includes MotionCreatorPro 2^{TM} suite of graphical installation, PID tuning, and data sampling for diagnostic programs
- Supports Windows® 2000/XP/Vista

1.2 Specifications

	Item	Description
System	Bus Type for PCI board	PCI Rev. 2.2, 33MHz
	Bus width for PCI	32-bit
	Bus Voltage	3.3V, 5V
	Memory usage	2M x 32 bit
	IRQ on PCI board	Assigned by PCI controller
General Specifications	Operating temperature	0°C to 55°C
	Storage temperature	-20°C to 80°C
	Humidity	5 to 95%, non-condensing
	Power Consumption	+3.3V @ 0.8 A typical
		+5V @ 0.8 A typical
		+/-12V @ 0.5 A typical
DSP	Type	TI TMS320C6711D
	Clock	250 MHz
	DSP performance	1500 MFLOPS
Board Interface	Major Connector	One SCSI VHDCI 68 P for PCI-8253
		Two SCSI VHDCI 68 P for PCI-8256
Servo Control	Maximum available axes	3/6 Axes for PCI-8253/6
	Analog Output command	±10 volts, resolution: 16 bit
	Analog Output signal type	Differential / Single-end
	Maximum Servo update rate	150 μs PID-FF for PCI-8253
		300 μs PID-FF for PCI-8256
	PID (Kp, Ki and Kd) gains	0 to 32,767
	Feed forward (Aff, Vff) gains	0 to 32,767
	Velocity feedback (Kv) gain	0 to 32,767
	Position range	32 bit
	Velocity range	32 bit
	Acceleration / Deceleration range	32 bit
	Encoder Input frequency	20 MHz @ 4x AB
	Encoder Input Interface	±12 volts, TTL compatible
	Encoder Filter	Yes

Table 1-1: Specifications

	Item	Description
Dedicated I/O	Motion I/O	Plus / Minus End Limit for each axis
		Zero-position for each axis
	Driver I/O (per axis)	Servo ON
		In-Position
		Zero-speed
		Alarm
		Error counter clear
	Others	Pulser Input (A/B/Index phase)
Analog Input	Number of inputs	Up to 6, multiplexed, single-ended
	Input coupling	DC
	Voltage range (programmable)	±10 V
	Bandwidth	100 kHz
	Resolution	14 bits, no missing codes
	All ranges	±1 mV for ±10 V input
	Maximum working voltage	±15 V
Analog outputs	Number of outputs	Up to 6, differential
	Output coupling	DC
	Voltage range	±5 V, ±10 V (Gain programmable)
	Output current	±50 mA (Typ.)
	Resolution	16 bits, no missing codes
	Monotonicity	Guaranteed
	Absolute accuracy	± 1.5LSB
	Protection	Short-circuit to ground
	Settling Time	15 μs , full-scale step
General purposed I/O		4 isolated DI/4 DO for PCI-8253
		8 isolated DI/8 DO for PCI-8256

Table 1-1: Specifications

	Item	Description
Motion Functions	Motion velocity profile	Trapezoidal & S-Curve
	Single motion	Jog move
		Point to Point motion
		Position / Speed Override
		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
	Synchronous Move	3 / 6 Axes for PCI-8253/PCI-8256
Application Functions	Gantry function	
	E-Gear function	
	Data sampling	
	System error check	Watchdog timer
Interrupt	During operation stop During alarms, etc.	Possible to select conditions where interrupt occurs
		Yes
Trigger Function	Type	Differential
	Number of outputs	Up to 4 (2 faster channels / 2 slower channels, supports encoders 0 and 3)
	Output low voltage	0. 5 V (Max.)
	Output high voltage	2.5 V (Min.)
	Polarity	Programmable, active-high or active-low
	Trigger output frequency	Faster: 1MHz Slower: 25KHz
	Min pulse width	200 ns (programmable pulse width)
	Support compared method	FIFO and Linear function
	Buffer size	15 points per trigger channel

Table 1-1: Specifications

Environmental Conditions

Item	Specification
Ambient TemperatureOperation	0 to 55°C
Ambient TemperatureStorage	-20 to 75°C
Ambient Humidity Operation	10 to 90% RH, avoid condensation
Ambient Humidity Storage	10 to 90% RH, avoid condensation
Noise resistance	Noise voltage 1500 V.P.P, noise frequency 25 to 60 Hz using noise simulator
Cooling method	Self-cooling

Table 1-2: Environmental Conditions

1.3 Supported Software

1.3.1 Programming Library

Windows 2000/XP/Vista DLLs are provided for PCI-8253/6 users programming their own applications. These function libraries are shipped with the board.

1.3.2 MotionCreatorPro 2^{TM}

This Windows-based utility is used to setup cards, motors, and systems. It can also aid in debugging hardware and software problems. It allows users to set I/O logic parameters to be loaded in their own program. This product is also bundled with the card.

Refer to Chapter 5 for more details.

1.4 Compatible Terminal Boards

ADLINK provides the DIN-825-J3A for Mitsubishi J3A servo driver I/O, such as alarm signals, zero-speed signals, analog monitor singals, and general purpose digital input signals. ADLINK also provides a general-purpose terminal board for wiring: the DIN-68S.

2 Installation

This chapter describes how to install the PCI-8253/6. Please follow these steps below:

- Check what you have (section 2.1)
- Check the PCB (section 2.2)
- Install the hardware (section 2.3)
- Install the software driver (section 2.4)
- Understand the I/O signal connections (chapter 3) and their operation (chapter 4)
- Understand the connector pin assignments (the remaining sections) and wiring the connections

2.1 Package Contents

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

- PCI-8253/6: DSP-based 3/6-Axis Analog
  Motion Control Card
- ADLINK All-in-one Compact Disc

The terminal board is an optional accessory. This would not be included in PCI-8253/6 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 for future shipment or storage.

2.2 PCI-8253/6 Outline Drawing

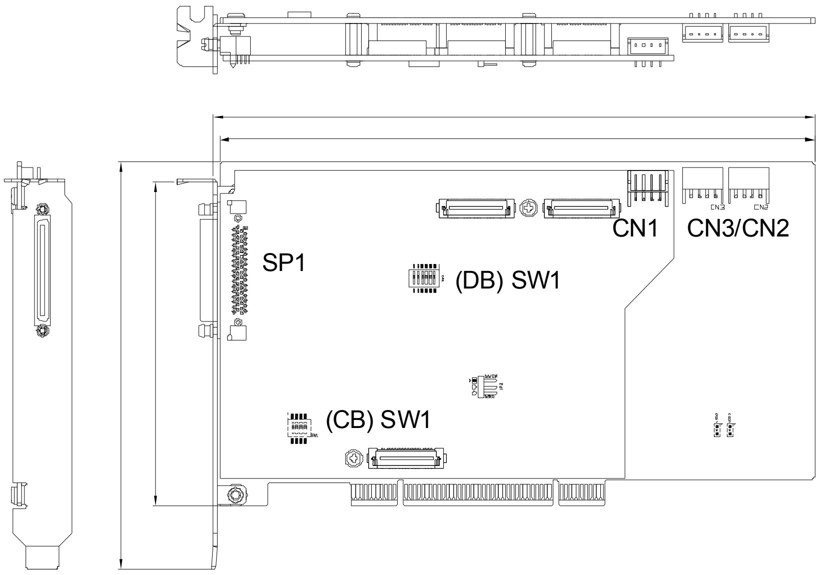


Figure 2-1: PCB Layout of the PCI-8253

SP1: Main signal connector (SCSI VHDCI 68 Pin)

CN2/CN3: DSP synchronous signal connector (4-pin)

CN1: Pulsar signal connector (4-pin)

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

(DB) SW1: DIP switch for differential/single-ended type of analog output signal selection

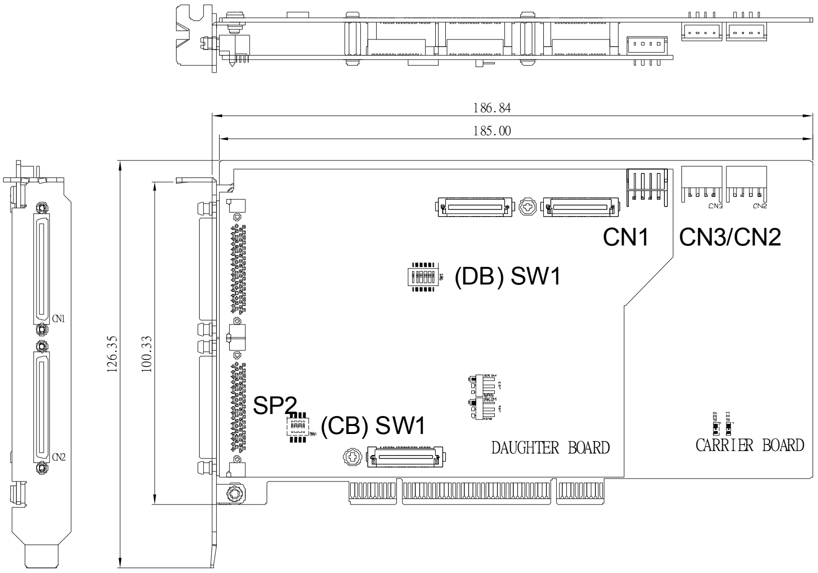


Figure 2-2: PCB Layout of the PCI-8256

SP1/SP2: Main signal connector (SCSI VHDCI 68-pin)

CN2/CN3: DSP synchronous signal connector (4-pin)

CN1: Pulsar signal connector (4-pin)

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

(DB) SW1: DIP switch for differential/single-ended type of analog output signal selection

2.3 PCI-8253/6 Hardware Installation

2.3.1 Hardware Configuration

The PCI-8253/6 is fully Plug and Play compliant. Therefore, memory allocation (I/O port locations) and the IRQ channel 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-8253/6 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-8253/6, 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.

2.3.4 Troubleshooting

If your system doesn’t boot or if you experience erratic operation with your PCI board in place, it’s most likely caused by an interrupt conflict (possibly an incorrect ISA setup). Once determined that the problem is more than a simple oversight, the general solution, 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.

2.4 Software Driver Installation

PCI-8253/6:

1. Autorun the ADLINK All-In-One CD. Choose Driver Installation -> Motion Control -> PCI-8253/6
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 SP1/SP2 Pin Assignments: Main Connector

SP1

No.	Name	I/O	Function of Axis	No.	Name	I/O	Function of Axis
1	AOUT1+	O	Analog output (+),(1)	35	AOUT1-	O	Analog output (-),(1)
2	AOUT2+	O	Analog output (+),(2)	36	AOUT2-	O	Analog output (-),(2)
3	AOUT3+	O	Analog output (+),(3)	37	AOUT3-	O	Analog output (-),(3)
4	AGND	SG	Analog ground	38	AGND	SG	Analog ground
5	AIN1	I	Analog input, (1)	39	AGND	SG	Analog ground
6	AIN2	I	Analog input, (2)	40	Rsv.	-	Reserved
7	AIN3	I	Analog input, (3)	41	Rsv.	-	Reserved
8	EA1+	I	Encoder A-phase (+),(1)	42	EA1-	I	Encoder A-phase (-),(1)
9	EB1+	I	Encoder B-phase (+),(1)	43	EB1-	I	Encoder B-phase (-),(1)
10	EZ1+	I	Encoder Z-phase (+),(1)	44	EZ1-	I	Encoder Z-phase (-),(1)
11	ALM1	I	Servo alarm,(1)	45	ORG1	I	Zero-Position, (1)
12	SVON1	O	Servo-ON, (1)	46	PEL1	I	Positive limit, (1)
13	ZSP1	I	ZeroSpeed 1	47	MEL1	I	Negative limit, (1)
14	TRG1	O	Trigger Output, (+)(1)	48	TRG1-	O	Trigger Output, (-)(1)
15	TRG2+	O	Trigger Output, (+)(2)	49	TRG2-	O	Trigger Output, (-)(2)
16	EA2+	I	Encoder A-phase (+),(2)	50	EA2-	I	Encoder A-phase (-),(2)
17	EB2+	I	Encoder B-phase (+),(2)	51	EB2-	I	Encoder B-phase (-),(2)
18	EZ2+	I	Encoder Z-phase (+),(2)	52	EZ2-	I	Encoder Z-phase (-),(2)
19	DOCOM	-	Digital output common	53	DICOM	-	Digital input common
20	ALM2	I	Servo alarm, (2)	54	ORG2	I	Zero-Position, (2)
21	SVON2	O	Servo-ON, (2)	55	PEL2	I	Positive limit, (2)
22	ZSP2	I	ZeroSpeed 2	56	MEL2	I	Negative limit, (2)
23	EA3+	I	Encoder A-phase (+),(3)	57	EA3-	I	Encoder A-phase (-),(3)
24	EB3+	I	Encoder B-phase (+),(3)	58	EB3-	I	Encoder B-phase (-),(3)
25	EZ3+	I	Encoder Z-phase (+),(3)	59	EZ3-	I	Encoder Z-phase (-),(3)
26	ALM3	I	Servo alarm,(3)	60	ORG3	I	Zero-Position, (3)
27	SVON3	O	Servo-ON, (3)	61	PEL3	I	Positive limit, (3)
28	ZSP3	I	ZeroSpeed 3	62	MEL3	I	Negative limit, (3)
29	DOCOM	-	Digital output common	63	IEMG	I	Emergency Stop
30	DOCOM	-	Digital output common	64	DICOM	-	Digital input common
31	EDO1	O	Digital Output, (1)	65	EDI1	I	Digital Input, (1)
32	EDO2	O	Digital Output, (2)	66	EDI2	I	Digital Input, (2)
33	EDO3	O	Digital Output, (3)	67	EDI3	I	Digital Input, (3)
34	EDO4	O	Digital Output, (4)	68	EDI4	I	Digital Input, (4)

Table 2-1: SP1 Pin Assignment

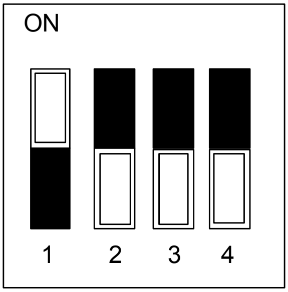
SP2

No.	Name	I/O	Function of Axis	No.	Name	I/O	Function of Axis
1	AOUT4+	O	Analog output (+),(4)	35	AOUT4-	O	Analog output (-),(4)
2	AOUT5+	O	Analog output (+),(5)	36	AOUT5-	O	Analog output (-),(5)
3	AOUT6+	O	Analog output (+),(6)	37	AOUT6-	O	Analog output (-),(6)
4	AGND	SG	Analog ground	38	AGND	SG	Analog ground
5	AIN4	I	Analog input, (4)	39	AGND	SG	Analog ground
6	AIN5	I	Analog input, (5)	40	Rsv.	-	Reserved
7	AIN6	I	Analog input, (6)	41	Rsv.	-	Reserved
8	EA4+	I	Encoder A-phase (+),(4)	42	EA4-	I	Encoder A-phase (-),(4)
9	EB4+	I	Encoder B-phase (+),(4)	43	EB4-	I	Encoder B-phase (-),(4)
10	EZ4+	I	Encoder Z-phase (+),(4)	44	EZ4-	I	Encoder Z-phase (-),(4)
11	ALM4	I	Servo alarm, (4)	45	ORG4	I	Zero-Position, (4)
12	SVON4	O	Servo-ON, (4)	46	PEL4	I	Positive limit, (4)
13	ZSP4	I	ZeroSpeed 4	47	MEL4	I	Negative limit, (4)
14	TRG3+ / OUT2+	O	Trigger Output, (+)(3) / Pulse Output, (+)(2)	48	TRG3- / OUT2-	O	Trigger Output, (-)(3) / Pulse Output, (-)(2)
15	TRG4+ / DIR2+	O	Trigger Output, (+)(4) / Pulse DIR., (+)(2)	49	TRG4- / DIR2-	O	Trigger Output, (-)(4) / Pulse DIR., (-)(2)
16	EA5+	I	Encoder A-phase (+),(5)	50	EA5-	I	Encoder A-phase (-),(5)
17	EB5+	I	Encoder B-phase (+),(5)	51	EB5-	I	Encoder B-phase (-),(5)
18	EZ5+	I	Encoder Z-phase (+),(5)	52	EZ5	I	Encoder Z-phase (-),(5)
19	DOCOM	-	Digital output common	53	DICOM	-	Digital input common
20	ALM5	I	Servo alarm,(5)	54	ORG5	I	Zero-Position, (5)
21	SVON5	O	Servo-ON, (5)	55	PEL5	I	Positive limit, (5)
22	ZSP5	I	ZeroSpeed 5	56	MEL5	I	Negative limit, (5)
23	EA6+	I	Encoder A-phase (+),(6)	57	EA6-	I	Encoder A-phase (-),(6)
24	EB6+	I	Encoder B-phase (+),(6)	58	EB6-	I	Encoder B-phase (-),(6)
25	EZ6+	I	Encoder Z-phase (+),(6)	59	EZ6-	I	Encoder Z-phase (-),(6)
26	ALM6	I	Servo alarm,(6)	60	ORG6	I	Zero-Position, (6)
27	SVON6	O	Servo-ON, (6)	61	PEL6	I	Positive limit, (6)
28	ZSP6	I	ZeroSpeed 6	62	MEL6	I	Negative limit, (6)
29	DOCOM	-	Digital output common	63	Rsv.	-	Reserved
30	DOCOM	-	Digital output common	64	DICOM	-	Digital input common
31	EDO5	O	Digital Output, (5)	65	EDI5	I	Digital Input, (5)
32	EDO6	O	Digital Output, (6)	66	EDI6	I	Digital Input, (6)
33	EDO7	O	Digital Output, (7)	67	EDI7	I	Digital Input, (7)
34	EDO8	O	Digital Output, (8)	68	EDI8	I	Digital Input, (8)

Table 2-2: SP2 Pin Assignment

2.6 (CB) SW1 Switch Setting for Card Index

The (CB) 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.



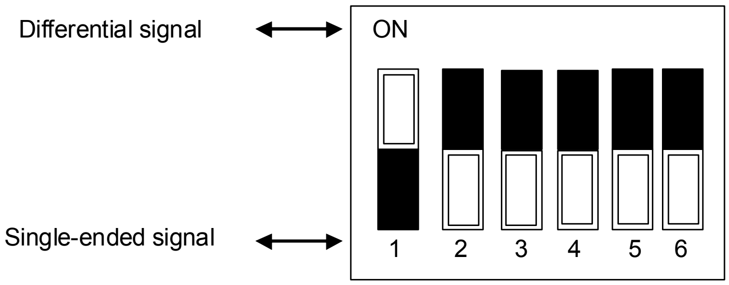
Card ID	Switch Setting (ON=0)
0	1111
1	1110
2	1101
3	1100
4	1011
5	1010
6	1001
7	1000
8	0111
9	0110
10	0101
11	0100
12	0011
13	0010
14	0001
15	0000

Table 2-3: SW1 Switch Settings

2.7 Daughter Board Switch (DB) SW1

(DB) SW1 is defined to select differential or single end function for "analog output".

(DB) SW1 Switch Diagram



2.8 CN1 Pulsar Connector

The CN1 connector is used to connect to a manual pluse generator.

CN1 Pin Assignment

Pin No	Signal Name
1	PDICOM
2	IPA
3	IPB
4	PDICOM

Table 2-4: CN1 Pin Assignment

2.9 CN2/CN3 DSP Sychronous Signal Connector

CN2/CN3 Pin Assignment

Pin No	Signal Name
1	SYN_PWR
2	STP
3	STA
4	DGND

Table 2-5: CN2/CN3 DSP Sychronous Signal Connector

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 cable between the PCI-8253/6 and any motor driver.

This chapter contains the following sections:

3.1 Analog Command Output Signals: AOUT

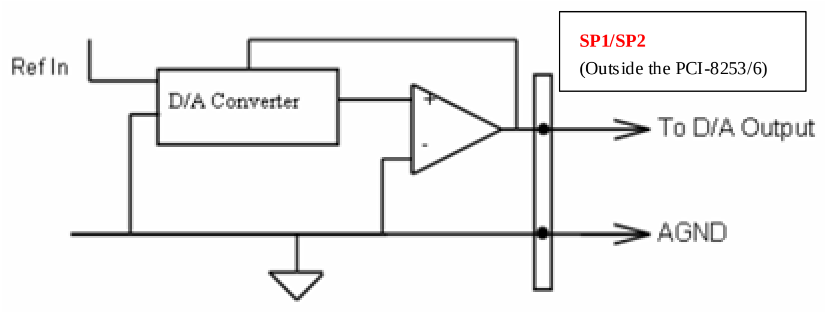
3.1.1 Single-ended Type Signal: AOUT+

There are 3/6 axes analog output signals on the PCI-8253/6. For each axis, the OUT+ signal is used to drive the voltage command and indicate the direction. The AOUT+ signals can also decide motor forward/backward direction by plus/minus voltage out. The Analog Output can drive between +10V to -10V and its resolution guarantees to ±1 mV. The following table shows all pulse output signals on SP1 and SP2.

SP1 Pin No.	Signal Name	Description	Axis #
1	**AOUT1+**	Analog Out Signal	1
2	**AOUT2+**	Analog Out Signal	2
3	**AOUT3+**	Analog Out Signal	3

SP2 Pin No.	Signal Name	Description	Axis #
1	**AOUT4+**	Analog Out Signal	4
2	**AOUT5+**	Analog Out Signal	5
3	**AOUT6+**	Analog Out Signal	6

The following wiring diagram is for Analog out signals of axis.



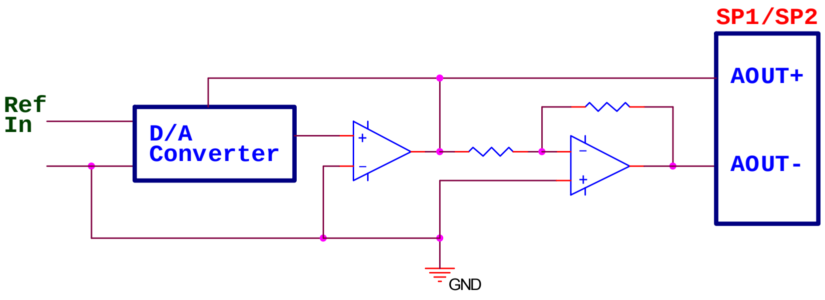
3.1.2 Differential Type Signals: AOUT+ / AOUT-

There are 3/6 axes differential analog output signals on the PCI-8253/6. For each axis, the AOUT signal is used to drive the voltage command and indicate the direction. The AOUT signals can also decide motor forward/backward direction by plus/minus voltage output. The analog output can drive between +10V to -10V and its resolution guarantees to 1mV. The following table shows all analog output signals in differential type on SP1 and SP2.

SP1 Pin No.	Signal Name	Description	Axis #
1	AOUT1+	Analog Out Signal +	1
35	AOUT1-	Analog Out Signal -	1
2	AOUT2+	Analog Out Signal +	2
36	AOUT2-	Analog Out Signal -	2
3	AOUT3+	Analog Out Signal +	3
37	AOUT3-	Analog Out Signal -	3

SP2 Pin No.	Signal Name	Description	Axis #
1	AOUT4+	Analog Out Signal +	4
35	AOUT4-	Analog Out Signal -	4
2	AOUT5+	Analog Out Signal +	5
36	AOUT5-	Analog Out Signal -	5
3	AOUT6+	Analog Out Signal +	6
37	AOUT6-	Analog Out Signal -	6

The following wiring diagram is for analog out signals of the axis.



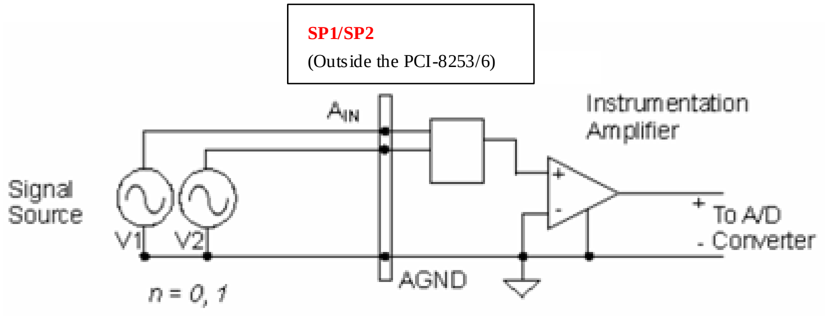
3.2 Analog Input Signals: AIN

There are 3/6 axes analog Output signals on the PCI-8253/6. For each axis, an AIN signal is used to receive the feedback voltage direction. The Analog Input can indicate the voltage level between +10V to -10V and its resolution guarantees 14 bits with no missing code. The following table shows all pulse output signals on SP1 and SP2.

SP1 Pin No.	Signal Name	Description
1	**AIN1**	Analog Input Signal
2	**AIN2**	Analog Input Signal
3	**AIN3**	Analog Input Signal

SP2 Pin No.	Signal Name	Description
1	**AIN4**	Analog Input Signal
2	**AIN5**	Analog Input Signal
3	**AIN6**	Analog Input Signal

The following wiring diagram is for Analog out signals of axis.



3.3 Trigger Pulse Output Signals: TRG+, TRG-

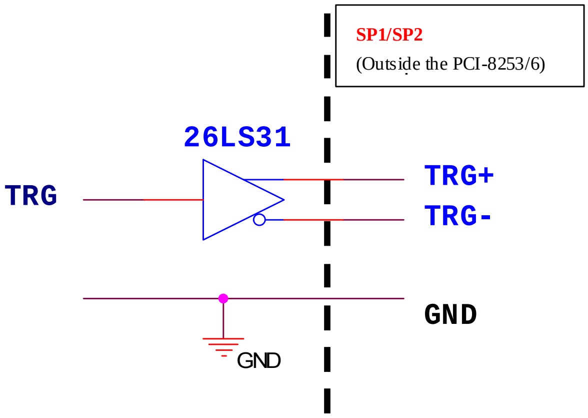
There are 2/4 axes trigger output signals on the PCI-8253/6. For each axis, one pair of TRG differential signals are used to transmit the trigger singal with pulse train type. Each signal consists of a pair of differential signals. For example, TRG1 consists of TRG1+ and TRG1- signals. The following table shows all trigger output signals on SP1 and SP2.

SP1 Pin No.	Signal Name	Description	Axis #
14	TRG1+	Trigger signal (+)	1
48	TRG1-	Trigger signal (-)	1
15	TRG2+	Trigger signal (+)	2
49	TRG2-	Trigger signal (-)	2

SP2 Pin No.	Signal Name	Description	Axis #
14	TRG3+	Trigger signal (+)	3
48	TRG3-	Trigger signal (-)	3
15	TRG4+	Trigger signal (+)	4
49	TRG4-	Trigger signal (-)	4

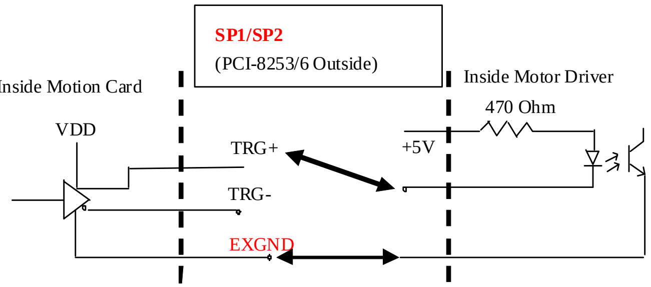
The **default** setting of TRG+ and TRG- is set to differential line driver mode.

The following wiring diagram is for TRG+ and TRG- signals of the axis.



Non-differential type wiring example:

Suggest Usage: See the following figure. Choose TRG+ to connect to driver's TRG



Warning: The sink current must not exceed 20mA or the 26LS31 will be damaged!

3.4 Encoder Feedback Signals: EA, EB and EZ

The encoder feedback signals include EA, EB, and EZ. Every axis has six pins for three differential pairs of phase-A (EA), phase-B (EB), and index (EZ) inputs. EA and EB are used for position counting, and EZ is used for zero position indexing. All relative signal names, pin numbers, and axis numbers are shown in the following tables:

SP1

SP1 Pin No	Signal Name	Axis #	SP1 Pin No	Signal Name	Axis #
7	EA1+	1	41	EA1-	1
8	EB1+	1	42	EB1-	1
16	EA2+	2	50	EA2-	2
17	EB2+	2	51	EB2-	2
23	EA3+	3	57	EA3-	3
24	EB3+	3	58	EB3-	3

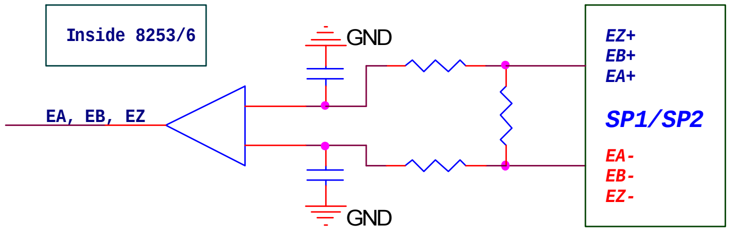
SP1 Pin No	Signal Name	Axis #	SP1 Pin No	Signal Name	Axis #
9	EZ1+	1	43	EZ1-	1
18	EZ2+	2	52	EZ2-	2
25	EZ3+	3	59	EZ3-	3

SP2

SP2 Pin No	Signal Name	Axis #	SP2 Pin No	Signal Name	Axis #
7	EA4+	4	41	EA4-	4
8	EB4+	4	42	EB4-	4
16	EA5+	5	50	EA5-	5
17	EB5+	5	51	EB5-	5
23	EA6+	6	57	EA6-	6
24	EB6+	6	58	EB6-	6

SP2 Pin No	Signal Name	Axis #	SP2 Pin No	Signal Name	Axis #
9	EZ4+	4	43	EZ4-	4
18	EZ5+	5	52	EZ5-	5
25	EZ6+	6	59	EZ6-	6

The input circuit of the EA, EB, and EZ signals is shown as follows:

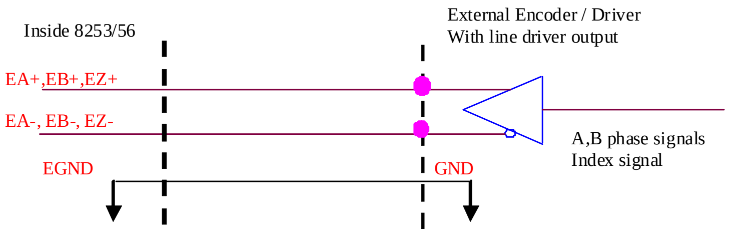


Please note that the voltage across each differential pair of encoder input signals (EA+, EA-), (EB+, EB-), and (EZ+, EZ-) have ±7V common mode range. Therefore, the output current must be observed when connecting to the encoder feedback or motor driver feedback as not to over drive the source. The differential signal pairs are converted to digital signals EA, EB, and EZ; then retrieve from the DSP side.

Below are examples of connecting the input signals with an external circuit. The input circuit can be connected to an encoder or motor driver if it is equipped with: (1) a differential line driver or (2) an open collector output.

Connection to Line Driver Output

To drive the PCI-8253/6 encoder input, the driver output must provide at least 0.2V across the differential pairs. The case grounds of both sides must be tied together. The maximum frequency is 5Mhz or more depends on wiring distance and signal conditioning.



For more operation information on the encoder feedback signals, refer to section 4.4.

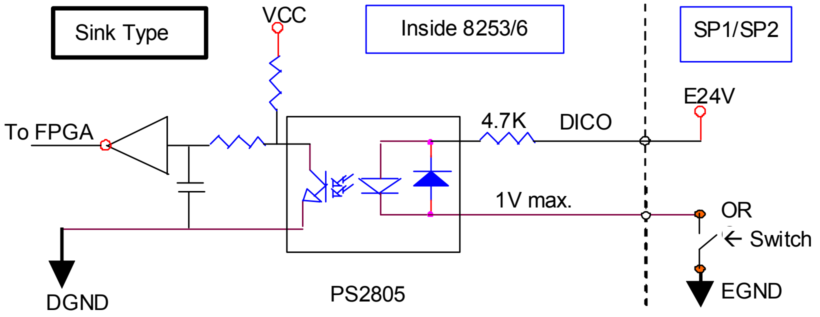
3.5 Origin Signal: ORG

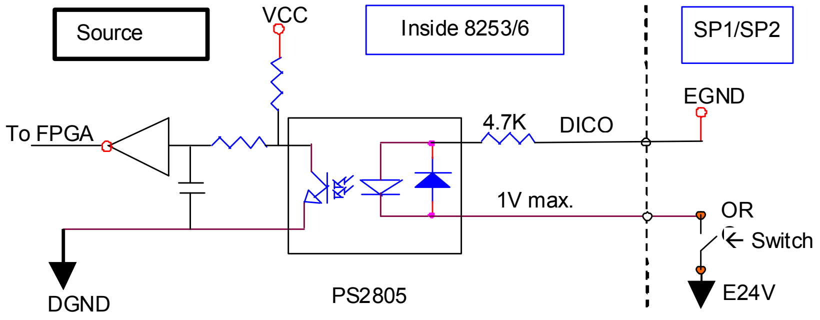
The origin signals (ORG1~ORG6) are used as input signals for the origin of the mechanism. The following table lists signal names, pin numbers, and axis numbers:

SP1 Pin No	Signal Name	Axis #	SP2 Pin No	Signal Name	Axis #
45	ORG1	1	45	ORG4	4
54	ORG2	2	54	ORG5	5
60	ORG3	3	60	ORG6	6

The input circuit of the ORG signals are shown below. Usually, a limit switch is used to indicate the origin on one axis. The specifications of the limit switch should have contact capacity of +24V @ 6mA minimum. An internal filter circuit is used to filter out any high frequency spikes, which may cause errors in the operation

.





When the motion controller is operated in the home return mode, the ORG signal is used to inhibit the control output signals (OUT and DIR).

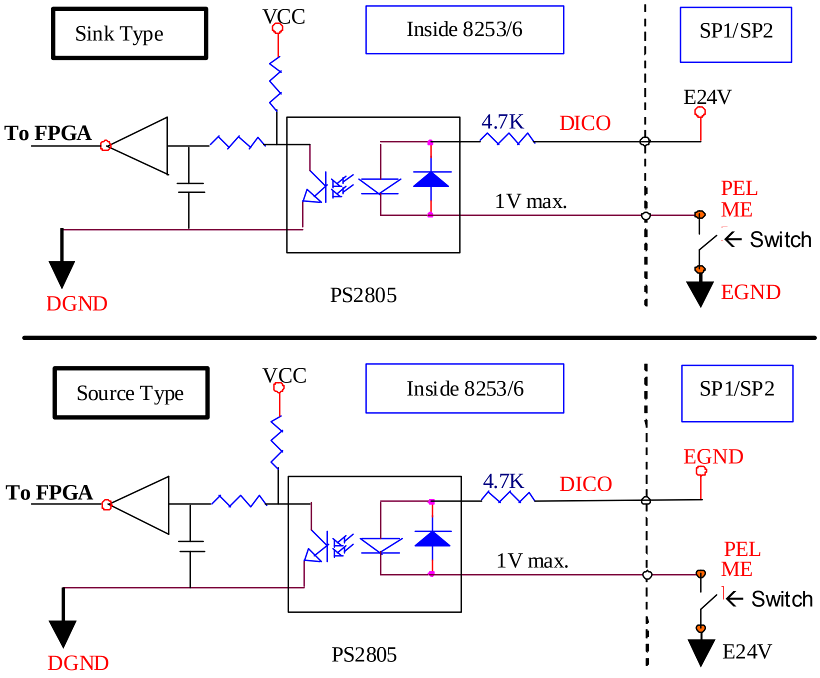
3.6 End-Limit Signals: PEL and MEL

There are two end-limit signals PEL and MEL for each axis. PEL indicates the end limit signal is in the plus direction and MEL indicates the end limit signal is in the minus direction. The signal names, pin numbers, and axis numbers are shown in the table below:

SP1 Pin No	Signal Name	Axis #	SP1 Pin No	Signal Name	Axis #
46	PEL1	1	47	MEL1	1
55	PEL2	2	56	MEL2	2
61	PEL3	3	62	MEL3	3

SP2 Pin No	Signal Name	Axis #	SP2 Pin No	Signal Name	Axis #
46	PEL4	4	47	MEL4	4
55	PEL5	5	56	MEL5	5
61	PEL6	6	62	MEL6	6

A circuit diagram is shown below. The external limit switch should have a contact capacity of +24V @ 6mA minimum. Either ‘A-type’ (normal open) contact or ‘B-type’ (normal closed) contact switches can be used.

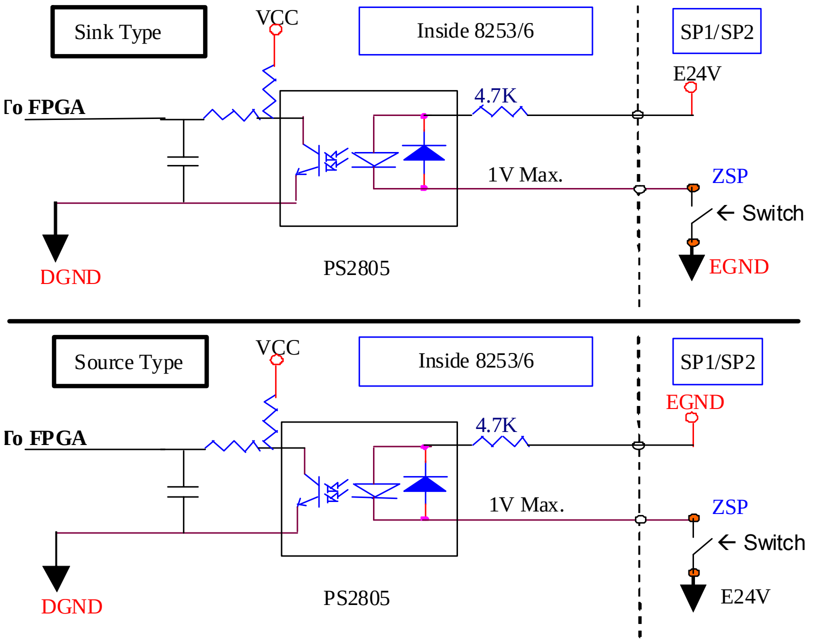


3.7 Zero Speed Signal: ZSP

The zero speed signal ZSP from a servomotor driver indicates zero speed. It is used to set the input range of the zero speed signals. The signal names, pin numbers, and axis numbers are shown in the table below:

SP1 Pin No	Signal Name	Axis #	SP2 Pin No	Signal Name	Axis #
13	ZSP1	1	13	ZSP4	4
22	ZSP2	2	22	ZSP5	5
28	ZSP3	3	28	ZSP6	6

The input circuits of the INP signals are shown in the diagram below:



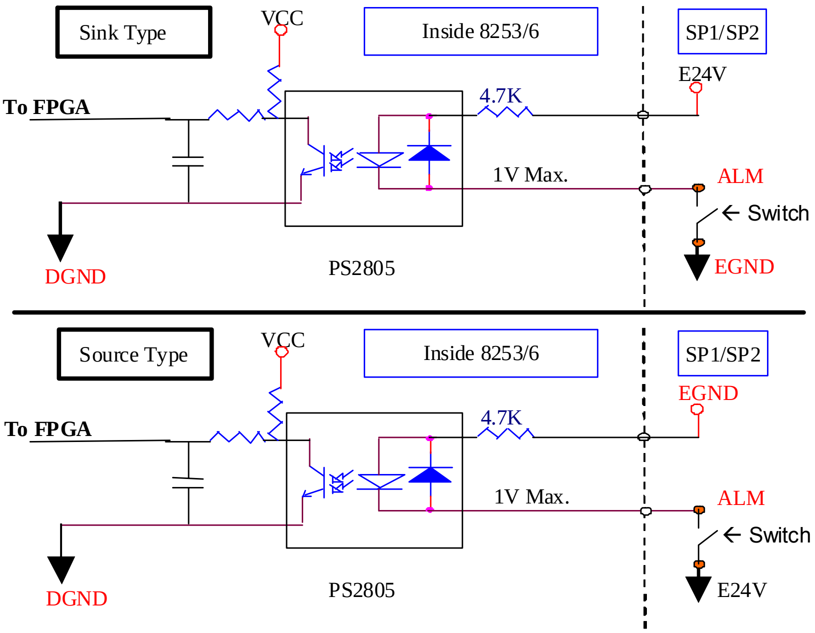
The in-position signal is usually generated by the servomotor driver and is ordinarily an open collector output signal. An external circuit must provide at least 6mA current sink capabilities to drive the ZSP signal.

3.8 Alarm Signal: ALM

The alarm signal ALM is used to indicate the alarm status from the servo driver. The signal names, pin numbers, and axis numbers are shown in the table below:

SP1 Pin No	Signal Name	Axis #	SP2 Pin No	Signal Name	Axis #
11	ALM1	1	11	ALM4	4
20	ALM2	2	20	ALM5	5
26	ALM3	3	26	ALM6	6

The input circuits of the ALM signals are shown in the diagram below:

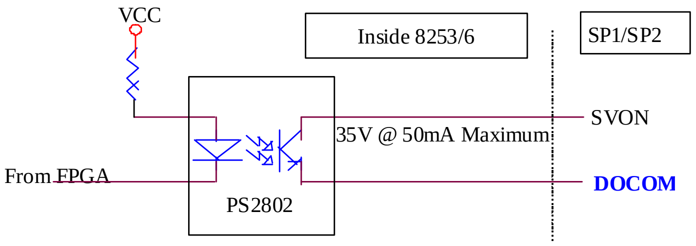


3.9 Servo ON Signal: SVON

The SVON signal can be used as a servomotor-on control or general purpose output signal. The signal names, pin numbers, and axis numbers are shown in the following table:

SP1 Pin No	Signal Name	Axis #	SP2 Pin No	Signal Name	Axis #
12	SVON1	1	12	SVON4	4
21	SVON2	2	21	SVON5	5
27	SVON3	3	27	SVON6	6

The output circuit for the SVON signal is shown below:

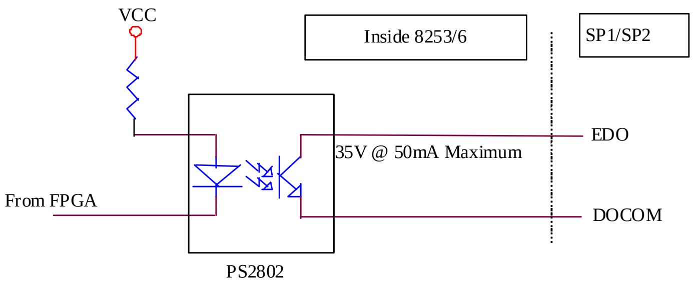


3.10 General-purpose Digital Output Signals: EDO

The PCI-8253/6 provides 4/8 general-purpose output channels: EDO1 to EDO8. These general-purpose output channels are located on SP1 and SP2. The signal names, pin numbers, and axis numbers are shown below:

SP1 Pin No	Signal Name	SP2 Pin No	Signal Name
31	EDO1	31	EDO5
32	EDO2	32	EDO6
33	EDO3	33	EDO7
34	EDO4	34	EDO8

The EDO signal wiring diagram is shown below:

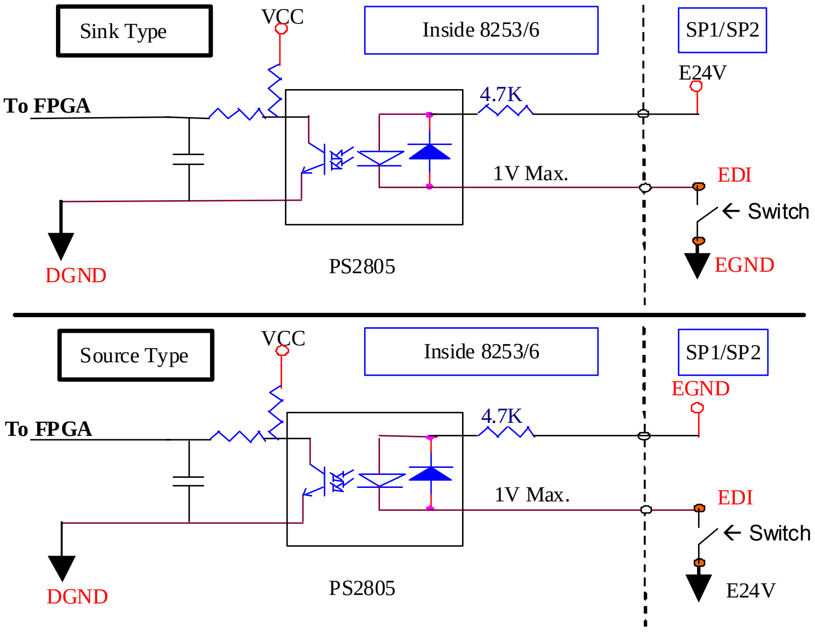


3.11 General-purpose Digital Input Signals: EDI

The PCI-8253/6 provides 4/8 general-purpose input channels: EDI1 to EDI8. These general-purpose input channels are located on SP1 and SP2. The signal names, pin numbers, and axis numbers are shown below:

SP1 Pin No	Signal Name	SP2 Pin No	Signal Name
65	EDI1	65	EDI5
66	EDI2	66	EDI6
67	EDI3	67	EDI7
68	EDI4	68	EDI8

The following wiring diagram is of the EDI signal:



4 Operation Theory

This chapter describes the detail operation of the motion controller card. Contents of the following sections are as follows:

4.1 Classifications of Motion Controller

When motor/stepper control first started, motion control was widely discussed instead of motor control. Motor control was separated into two layers: motor control and motion control. Motor control relates to PWM, power stage, closed loop, hall sensors, vector space, etc. Motion control relates to speed profile generating, trajectory following, multi-axes synchronization, and coordinating.

4.1.1 Analog Type Motion Control Interface

The interfaces between motion and motor control are changing rapidly. Early on, a voltage signal was used as a command to the motor controller. The amplitude of the signal shows how fast a motor is rotating and the time duration of the voltage change shows the speed of the motor acceleration. The voltage signal as a command to the motor driver is called “analog” motion controller. It is much easier to integrate into an analog circuit of motor controller; however noise is sometimes a big problem for this type of motion control. Also, to do positioning control of a motor, the analog motion controller must have a feedback signal with position information and use a closed loop control algorithm to make it possible. This increases the complexity of motion control and is not easy to use for a beginner.

4.1.2 Pulse Type Motion Control Interface

The second interface of motion and motor control is a pulse train type. As a digital world trend, pulse trains represent a new concept to motion control. The number of pulses illustrate how many steps a motor rotates, and the frequency of pulses illustrate how fast a motor runs. The time duration of frequency changes represent the acceleration rate of a motor. This interface makes a servo or stepper motor easier than an analog type for positioning applications, because it allows motion and motor control to be separated easier.

Each interface provides gains tuning. For analog position controllers, the control loops are built inside and gains are tuned from the controller. For pulse type position controllers, the control loops are built outside on the motor drivers and the gains are tuned on the drivers.

For operating more than one axis, motion control seems more important than motor control. In industrial applications, reliability is a very important factor. Motor driver vendors make good performance products and motion controller vendors make a powerful variety of motion software. Integrating the two products make this ideal.

4.1.3 Network Type Motion Control Interface

Recently, a new control interface was introduced--a network motion controller. In this new interface the command between motor driver and motion controller is neither an analog nor a pulse signal, it is a network packet containing position and motor information. This controller is more reliable than previous controllers because it is digitized and packetized. Because a motion controller must be real-time, the narrow must have real-time capacity at a cycle time below 1 mini-second. This means that non-commercial networks cannot do this job. A specific network is required, such as Mitsubishi SSCNET. The network may also be built with fiber optics to increase communication reliability.

4.1.4 Software Real-time Motion Control Kernel

For motion control kernel, there are three ways to accomplish it: DSP, ASIC, and software real-time.

A motion control system needs an absolute real-time control cycle. It order for the motor to run smoothly the calculation on the controller must provide a control data. The PC's computing power is often used to do this. A feedback counter card can simply be used and a voltage output or pulse output card to make it. This method is low cost but required extensive software effort. A real-time software is used to ensure real-time performance. This method increases the complexity of the system, but is the most flexible for professional motion control designers. Most of these methods are on NC machines.

4.1.5 ADLINK Softmotion DSP

ADLINK softmotion was implemented inside a high performance DSP that solves real-time software problem on computer and offers variety application functions to benefit you to realize your design conveniently. A DSP is a micro-processer itself and all motion control calculations can be performed on it. There is no real-time software problem because DSP has its own OS to arrange all the procedures. There is no interruption from other inputs or context-switching problems as seen in a Windows-based computer. Although it has such a perfect performance on real-time requirements, its calculation speed is not as fast as PC's CPU at this age. Besides, the software interfacing between DSP based controller's vendors and users are not easy to use. Some controller vendors provide some kind of assembly languages for users to learn and some controller vendors provide only a handshake documents for users to use. Both ways are not easy to use. A DSP based controller providse a better way for machine makers to develop they applications over software kernels. More features of ADLINK softmotion are described in Section 4.1.8.

4.1.6 ASIC Motion Control Kernel

An ASIC motion control kernel falls between software kernel and DSP kernel in terms of difficulty. All motion functions are done via the ASIC elimination all real-time problems. The ASIC requires parameters to be preset for motion control. ASIC motion control separates all system integration problems into 4 parts: motor driver's performance, ASIC outputting profile, vendor's software parameters to the ASIC, and users' command to vendor's software. It makes motion controller co-operated more smoothly between devices.

4.1.7 Comparison Table Of All Motion Control Types

	Analog	**Pulses**	**Network**
Price	**High	Low	**Normal
Signal Quality (refer to distance)	Good	Good	Best
Maintenance	Middle	Fair	Easy

**DSP or software real-time OS is needed

4.1.8 PCI-8253/6's Motion Controller Type

The PCI-8253/6 is a DSP based analog type motion controller. This card was made into five blocks: on-board DSP, ADLINK softmotion, FPGA for the interface, PCI card, and software motion library. The ADLINK softmotion easily utilizes for complex motion manipulation which involving jogging, point-to-point positioning, multiple axes synchronized motion, contouring, high-speed position comparison and so on. PCI-8253/6 also offers motion kernel customization service to met your requirement. Providing on faster encoder speeds, high speed position comparison, adjustable PID plus feed-forward closed-loop control algorithm and large size, 32-bit high-speed dual port ram usage, the PCI-8253/6 allows for high-speed servo control up to 20 million encoder counter per second and high-speed trigger pulse output up to 1 MHz. Other advantages of ADLINK softmotion technology is the servo update rates as low as 50 µs per axis.

The PCI-8253/6 also provides the programmable acceleration and deceleration to eliminate jerk and smooth velocity profile. For each axis, individual unlimited point table with 32 depth buffers can realize the seamless continuous move conveniently, which is also able to combine linear and arc segment.

The ADLINK softmotion has many kinds of choices for different application. It can be downloaded and updated on the PC side. Different DSP kernel has special motion functions. For standard package of PCI-8253/6, it has the most common motion control features inside. For others, please order the specific kernel as needed.

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

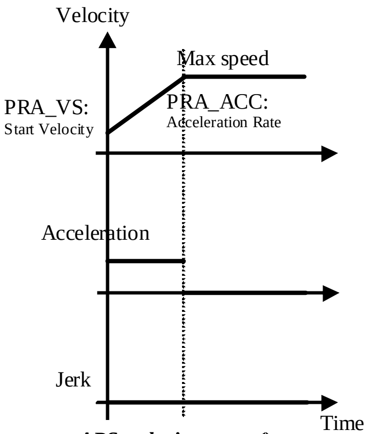
Veloctiy mode means the motion command is continuously outputting until a stop command is issued. The motor will run without a target position or desired distance unless it was stopped by other reasons. The output command profile will accelerate from a starting velocity to a specified maximum velocity. 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.

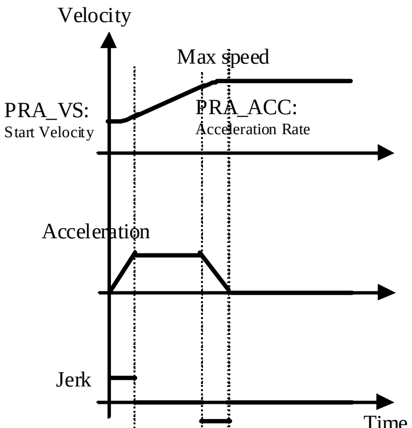
```
APS_set_axis_param(Axis_ID, AXS_Param_No,
     AXS_Param)
APS_velocity_move(Axis_ID, Max_speed)
```

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



APS_velocity_move () with T-curve pattern



APS_velocity_move () with S-curve pattern

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.

4.2.2 Single Axis P-to-P Motion

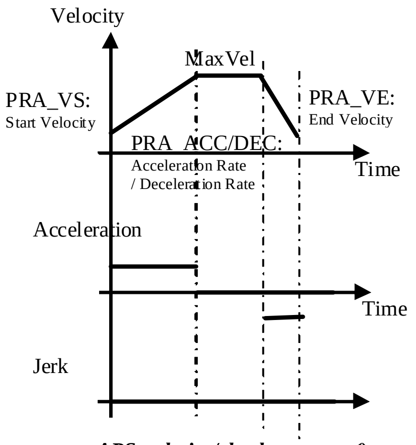
In this section, the following functions are discussed.

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

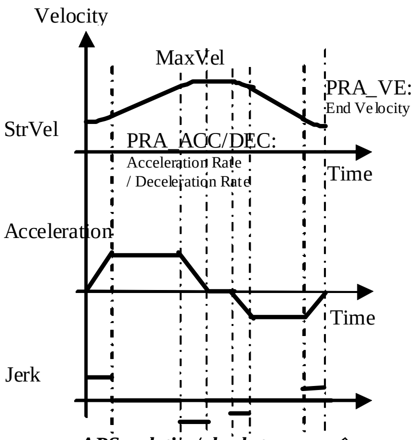
Single axis P-to-P motion functions will allow the axis to move a specified distance or move to a specified position. Above two functions are pretty straightforward. ‘Relative’ and ’absolute’ characterizes the function and provides information about the position method to achieve the target position. 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).



APS_relative/absolute_move () with T-curve pattern



APS_relative/absolute_move () with S-curve pattern

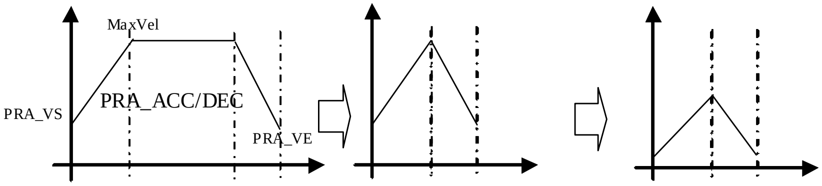
Note: Please refer to section 6.11 – “Definition table” to set the axis parameter.

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

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)

4.2.3 Linear Interpolation

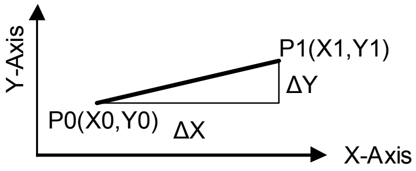
“Interpolation between multi-axes” means these axes start simultaneously, and reach their ending points at the same time. Linear means the ratio of speed of every axis is a constant value. In this section, the following functions are described.

```
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)
```

These two 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.

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)
```

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.

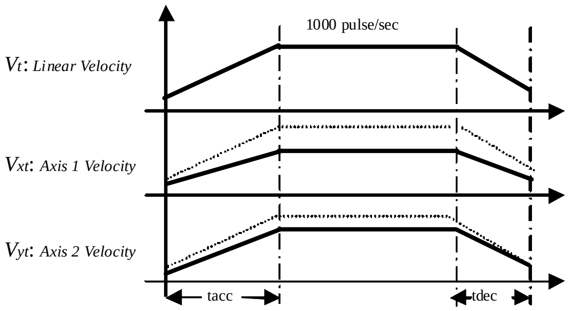
```
…
Ret = APS_absolute_linear_move ( Dimension,
     Axis_ID_Array, Position_Array,
     Max_Linear_Speed );
…
```

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-8253/6 inner, DSP will handle the velocity profile of first and second axis to match two axes interpolation motion requirement. Basically, PCI-8253/6 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.

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

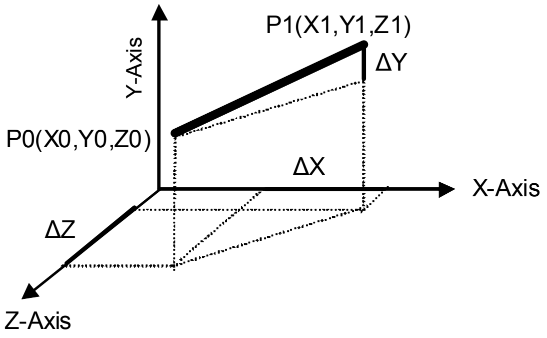
In which V_t is vector speed of linear interpolation motion; V_{xt} is speed along x direction (first axis); V_{yt} is speed along y direction (second axis).



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.

3 Axes Linear Interpolation

Any 3 of the 16 axes of a DSP-based 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)
```

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.

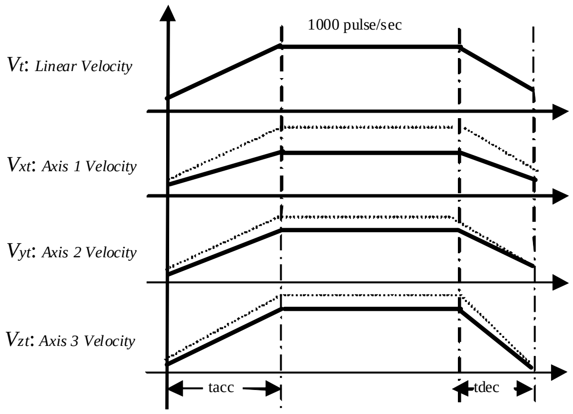
```
…
Ret = APS_absolute_linear_move (Dimension,
     Axis_ID_Array, Position_Array,
     Max_Linear_Speed );
…
```

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, inside the PCI-8253/6, the DSP will handle the velocity profile of first and second axis to match two axes interpolation motion requirement. Basically, PCI-8253/6 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.

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

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.

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.

4.2.4 Circular Interpolation

Any 2 of the 16 axes of a DSP-based analog 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

```
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)
```

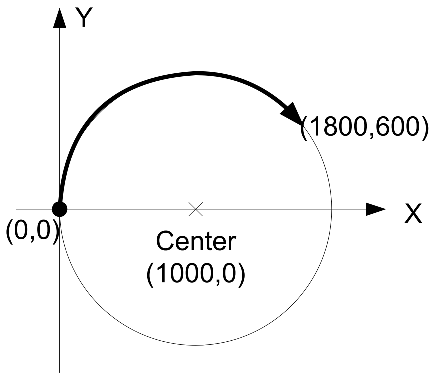
Example:

```
//…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)

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



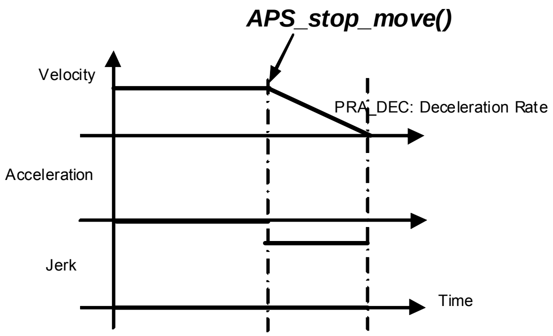
```
//…
APS_set_axis_param( Master_Axis_ID, PRA_CURVE, 1
      );  //Set S-curve
APS_set_axis_param( Master_Axis_ID, PRA_ACC,
      100000 ); //Set acceleration
APS_set_axis_param( Master_Axis_ID, PRA_DEC,
      100000 ); //Set deceleration
Ret = APS_absolute_arc_move( Dimension,
      Axis_ID_Array,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

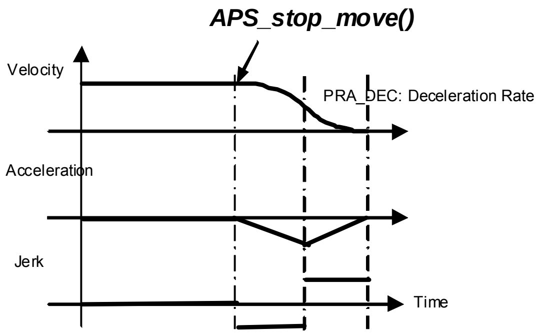
After whole parameter setting completed and then user is able to perform the circular interpolation function in absolute mode.

4.2.5 Speed Override

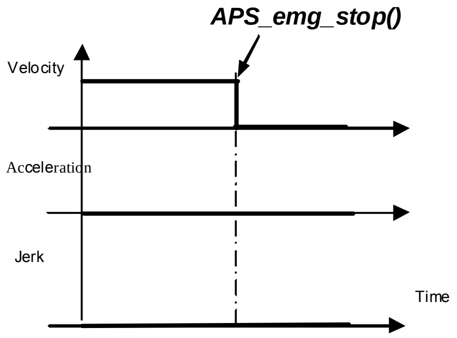
Speed override means that users can change command's speed during the operation of motion. User is easy to modify the speed profile during motion operation by re-executing the function **`APS_velocity_move`**().This function can be applied on other motion in position or velocity mode. If the running 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.



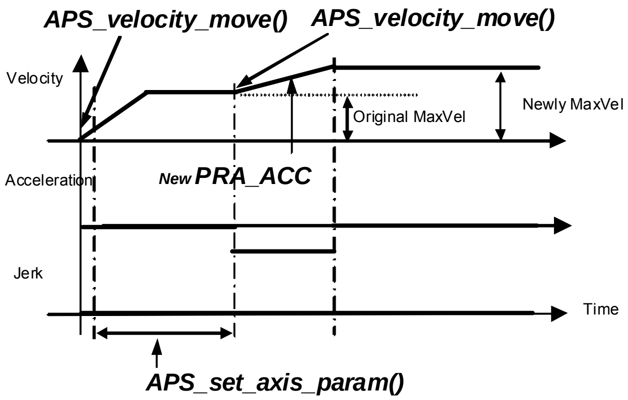


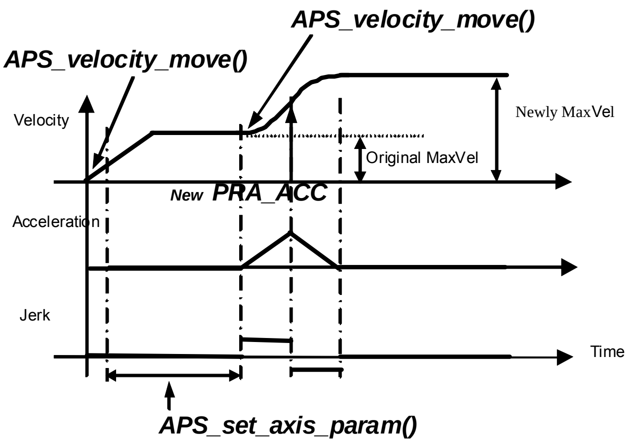
The function **APS_emg_stop**() stops the a specified 'Axis' immediately without deceleration. See diagram below.



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.



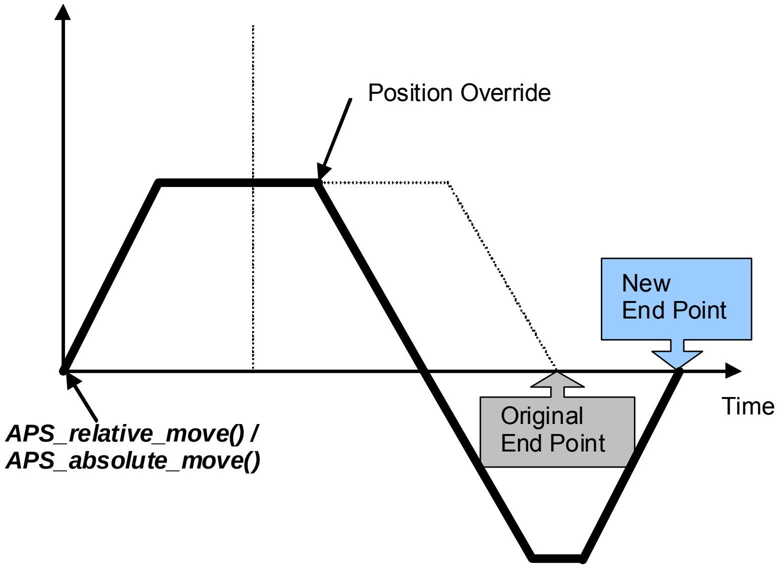


Use the same method to change the velocity during axis traveling with S-curve velocity profile.

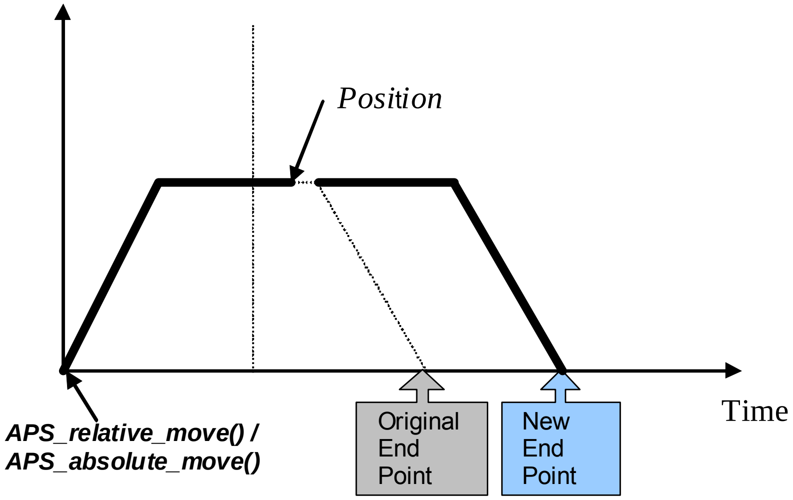
Note: All change speed on the fly function calls can be applied any time when an axis is moving but cannot be overridden by other motion modes like Jog, home, manual pulse generation, contour motion and G-code program. The axis motion **must be stopped** before switching to those modes mentioned above.

4.2.6 Position Override

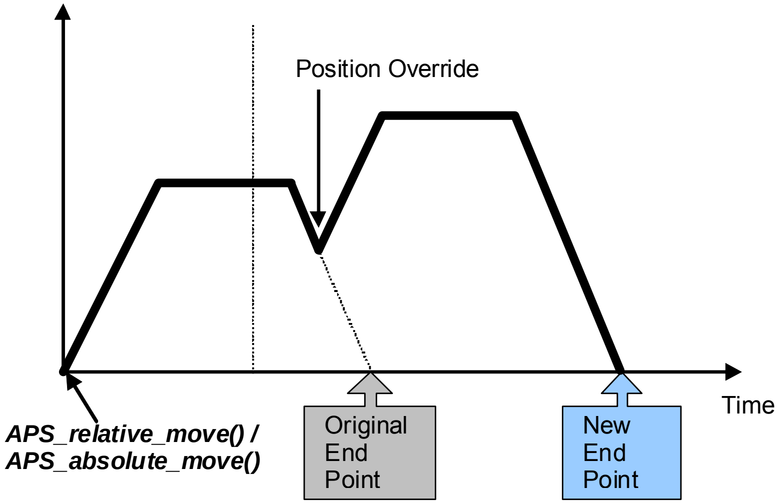
Position override means that when users issue a positioning command and want to change its target position during this operation. If the new target position is behind current position when override command is issued, the motor will slow down then reverse to new target position. If the new target position 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.



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.



Position
Time
APS_relative_move() /
APS_absolute_move()
Original End Point
New End Point



Position Override
Time
APS_relative_move() /
APS_absolute_move()
Original End Point
New End Point

4.3 Home Move

In this section, the following functions are discussed.

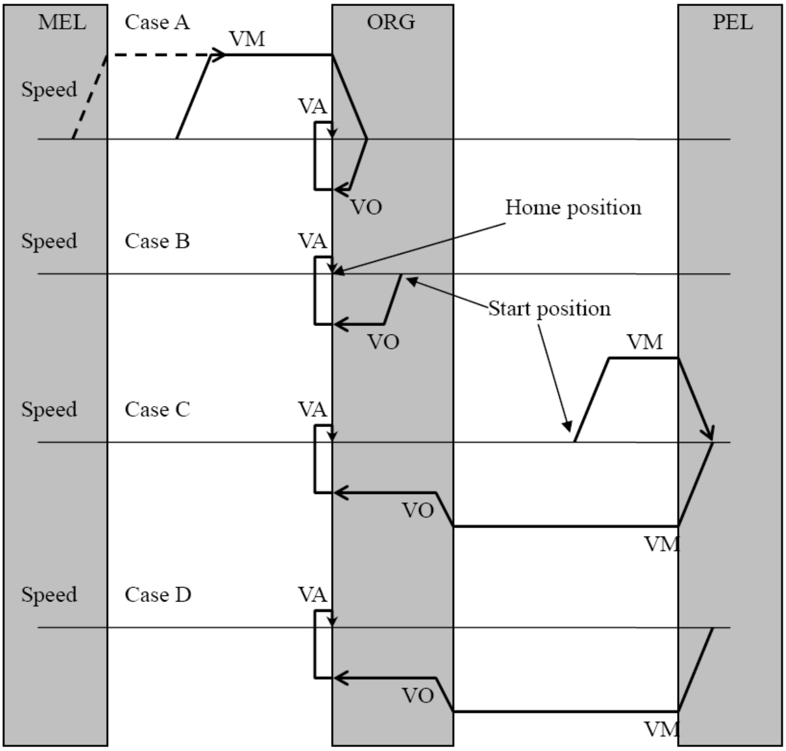
```
APS_home_move(Axis_ID)
```

The above function is used to set the home return mode. One Home modes are available in PCI-8253/6 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-8253/6, 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

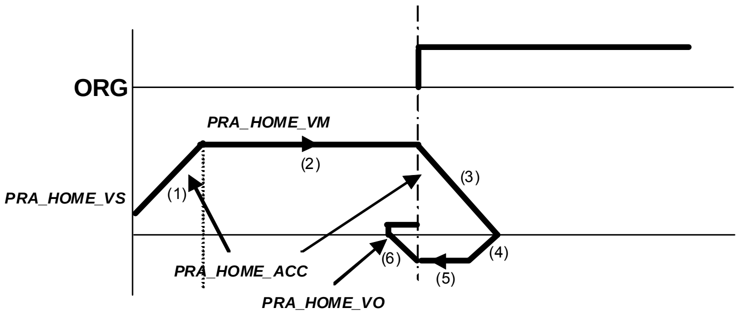
Table 4-1: Axis Parameter Settings for ORG Home Moves



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

Here is an example to briefly describe what does the motion behavior after executed the homing move function.

Case A



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 update cycle time until ORG turn ON, then stop and finish home return.

Example:

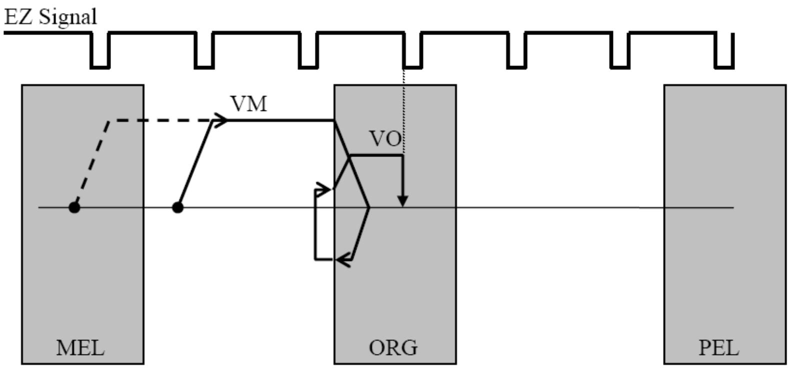
```
//Set homing parameters
APS_set_axis_param( Axis_ID, PRA_HOME_MODE, 0 );
      //Set home mode

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

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)
```

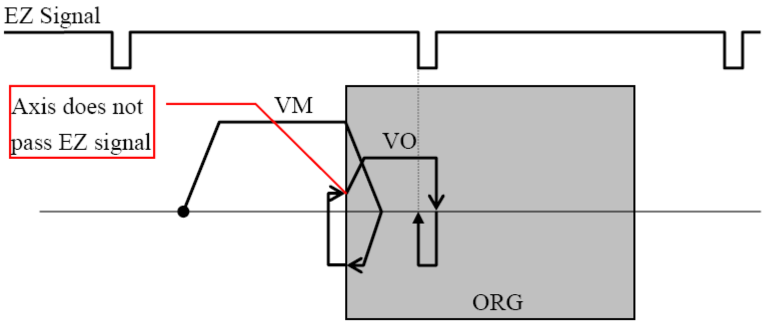
In addition to home moves using the ORG signal, this board also offers home moves via the index signal (EZ) as illustrated:

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



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.



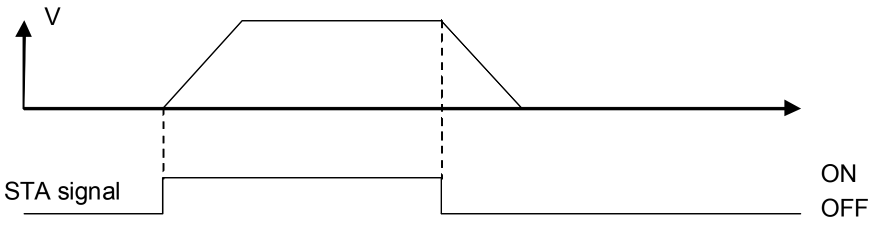
4.4 Jogging

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.

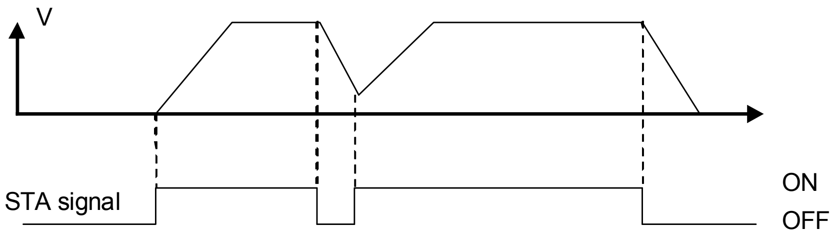
Free-running Mode:

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 ‘**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.



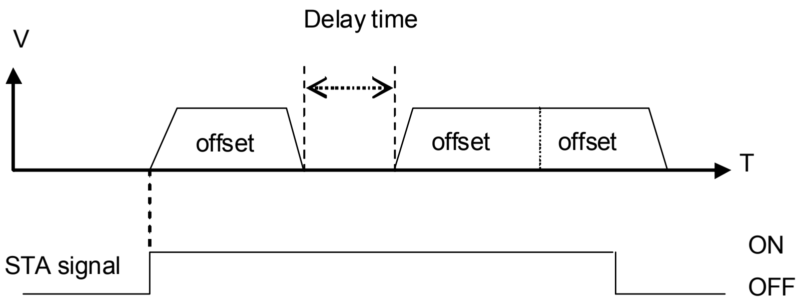
When the axis is decelerating and the STA signal is turned on again, the axis will re-accelerate to the maximum velocity.

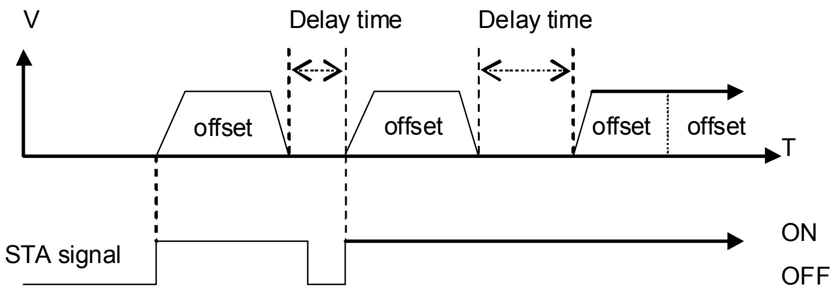


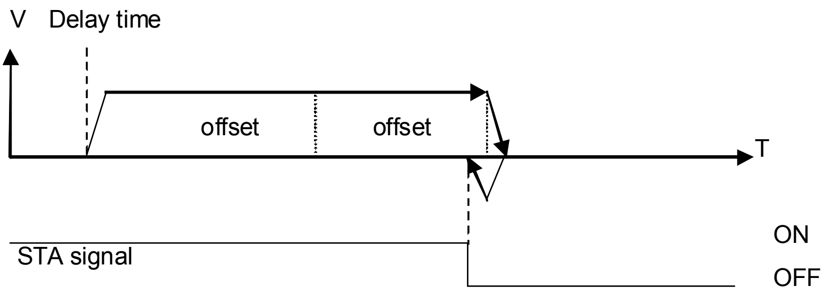
Step Mode:

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.

Operation flow	APS functions

Configure Jog parameters	APS_set_axis_param()
	APS_get_axis_param()
	APS_set_jog_param()
	APS_get_jog_param()

Switch axis to jog standby state	APS_jog_mode_switch();
Jog move operations	APS_jog_start()
Switch axis to normal state	APS_jog_mode_switch();

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)
…
```

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
```

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.

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.)

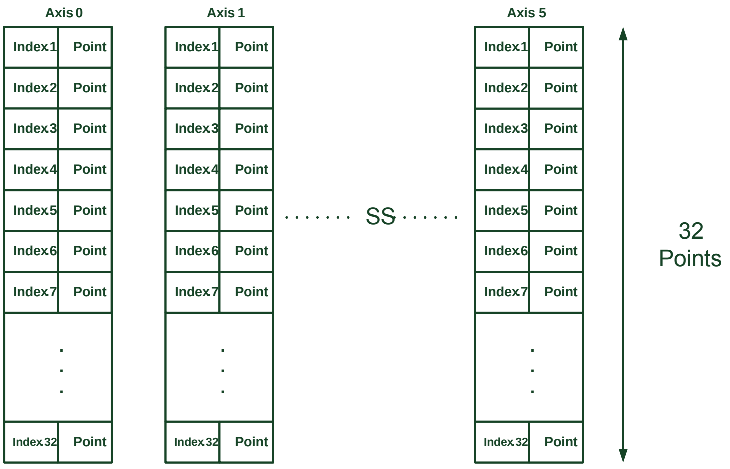
Bit ‘0’ decides the moving method to relative mode or absolute mode. Bit ‘2’ decides the trajectory moving following linear interpolation or circular interpolation method. Bit ‘0’ decides the moving method to relative mode or absolute mode. Bit ‘4’ decides the stop method that refers to ‘INP’ signal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

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-8253/6 allow 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'.

4.5.3 Point table execution

In this section, the following function is discussed.

```
APS_point_table_move(Dimension, *Axis_ID_Array,
    StartIndex, EndIndex)
```

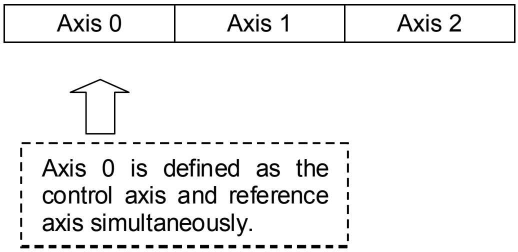
By using the **`APS_xxx_table_move`**() function, the point index is dedicated to identify the start point, next point, or end point which is operating in point table sequence. By using the aforementioned functions to pause the point table movement or set the point table movement to repeat, when the pause command is issued, it will not stop the current point but stop at the next point index. Further-more, 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 inter-polation 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, mean-ing it decides the behavior of interpolation. Therefore, the slave axis will follow the control axis to execute its movement corre-spondingly. It allows for several existing slave axes. For the refer-ence 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-tion, then you just need to program the related motion parameter for the reference axis.

For example, in Case A, if you need to perform the 3D linear interpolation with continuous moving by point table configuration, then the axis 0 forms the first element of the three axes array, following the rule that mentioned above. Axis 0 shall be defined to reference axis, and then the other two axes will follow the axis 0 parameter for their usage. Furthermore, the axis 0 is also the minor axis (with minor axis number within the three axes array), and therefore axis 0 also roles the control axis in this axes array. The axis 1 and axis 2 will process the interpolation move and follow the axis 0 accordingly.

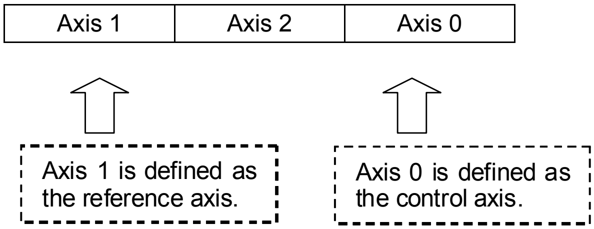
Case A



Axis 0 is defined as the control axis and reference axis simultaneously.

In the illustration below (Case B), axis 1 was the first element instead of axis 0 of a three axes array so that the axis 1 would be the reference axis in this array. In other way, axis 0 is still the minor axis in this array, and therefore it is still the control axis in this array.

Case B



Axis 1 is defined as the reference axis.

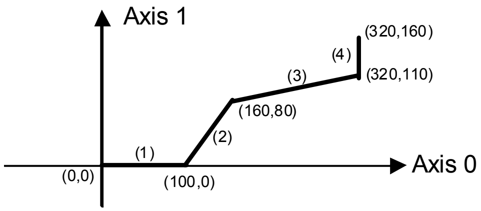
Axis 0 is defined as the control axis.

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:

Example: 2-D Linear move

The resulting 2-D trajectory is:



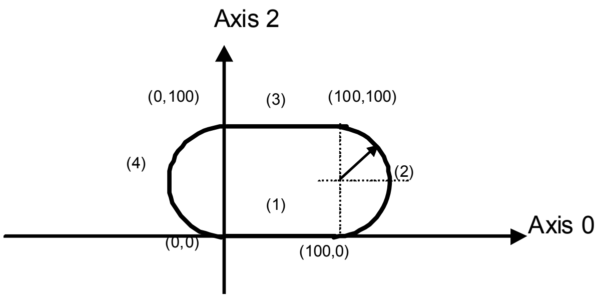
```
#include "type_def.h"
#include "APS_define.h"
#include "APS168.h"
#include "ErrorCodeDef.h"

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 );
...
```

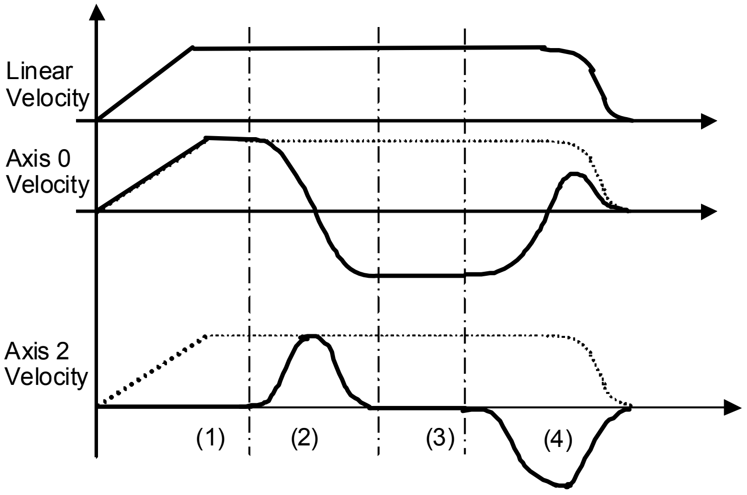
A 2-D arc trajectory is also able to add to the continuous motion trajectory in the point table.

Example: 2-D Linear Move

The resulting 2-D trajectory is:



The resulting velocity vs. time is:



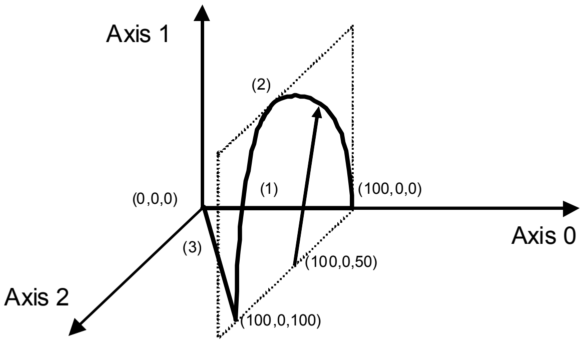
```
#include “type_def.h”
#include “APS_define.h”
#include “APS168.h”
#include “ErrorCodeDef.h”

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 );
…
```

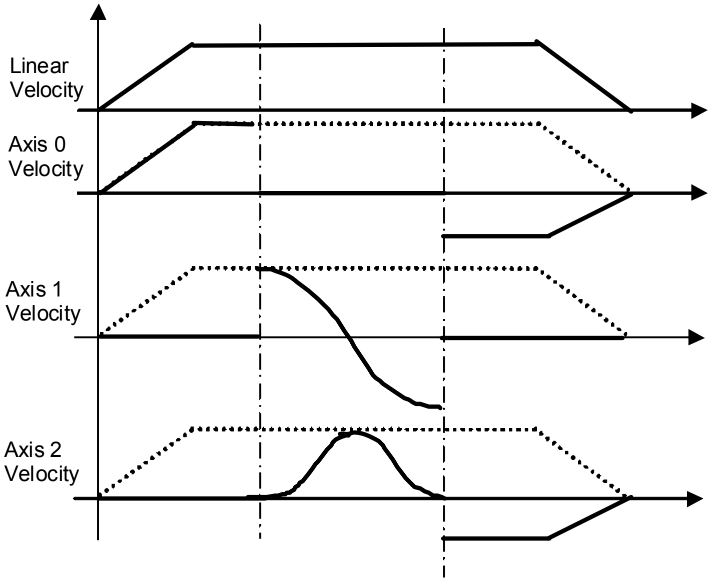
A 3-D trajectory is also able to add to the continuous motion trajectory in the point table.

Example: 2-D Linear Move

The resulting 3-D trajectory is:



The resulting velocity vs. time is:



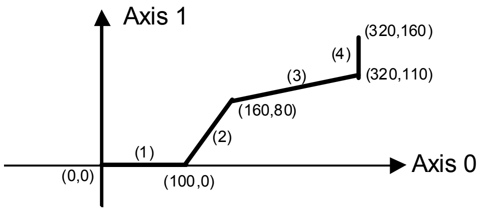
```
I32 ret;
POINT_DATA Point_AX0, Point_AX1, Point_AX2;
I32 Axis_ID_Array[3] = {0, 1, 2};

// ********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);
…
```

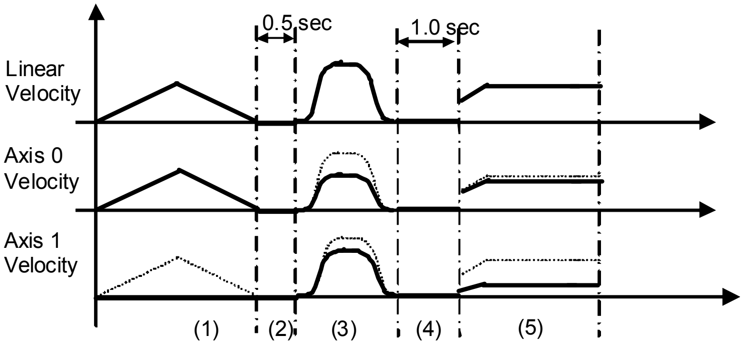
```
//Set and save parameter for AX1 and AX2

// *******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;
…
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:

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
```

```
//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 );
…
```

4.6 Motion Status and Related IO Monitoring

The PCI-8253/6 board has vast dedicated I/O 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. In addition, the motion status will be updated cyclically by motion status monitoring function within per servo update cycle – 300 us (/ 6 axes). 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: Velocity Feedback
- Section 4.6.3: Motion I/O Status
- Section 4.6.4: Motion Status

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 DSP-based analog motion controller - PCI-8253/6 controls servo drivers & motors via volt commands. For each servo update cycle (50 us per axis), the PCI-8253/6 sends a command to and receives a response from the servo driver via encoder feedback. 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.

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 servo update cycle (50 us per axis), the PCI-8253/6 sends a command to and receives a response from the servo driver via encoder feedback. 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.

4.6.2 Velocity Feedback

In this section, the following functions are discussed.

```
APS_get_command_velocity(Axis_ID, *Velocity)
APS_get_feedback_velocity(Axis_ID, *Velocity)
```

Get feedback velocity information

The DSP-based analog motion controller - PCI-8253/6 controls servo drivers & motors via volt command. For each servo update cycle (50 us per axis), the PCI-8253/6 sends a command to and receives a response from the servo driver via encoder feedback. Through command and response, an abundant amount of information is carried in and out, including position command and position feedback. Due to the closed-loop inside DSP and the position feedback information will be also converted to velocity feedback information to easy monitor the deviation between velocity command and response. The function call **`APS_get_feedback_velocity`**() will retrieve such information. The parameter '**`*Velocity`**' retrieves the current velocity feedback, which present motor speed from the servo driver.

Get command velocity information

For each servo update cycle (50 us per axis), the PCI-8253/6 sends a command to and receives a response from the servo driver. The function call **`APS_get_command_velocity`**() will retrieve command information which be calculated by DSP and the parameter '**`*Velocity`**' retrieves the next command, user is easy to monitor the command status and verify the generated profile by such function.

4.6.3 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`**().

EMG signal

Each PCI-8253/6 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

4.6.4 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

4.7 Driver Management

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

4.7.1 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.

4.8 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-8253/6, 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.

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

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

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.

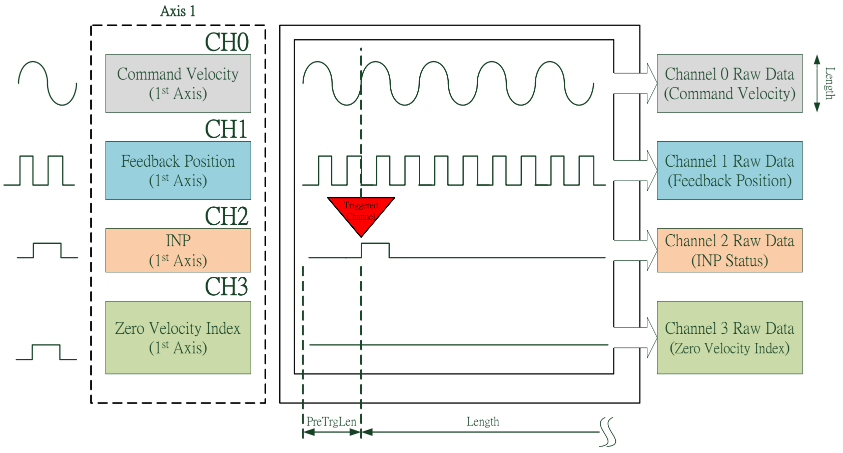
1. Configure trigger condition

   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.

2. Select the sampling source

After sampling condition configuration completion, user needs to decide what kinds date wants to monitor, in PCI-8253/6, ADLINK offers 14 sampling sources for monitoring and they are listed as the above table.


Axis 1
CH0
Command Velocity (1st Axis)
CH1
Feedback Position (1st Axis)
CH2
INP (1st Axis)
CH3
Zero Velocity Index (1st Axis)
Triggered Channel
Channel 0 Raw Data (Command Velocity)
Channel 1 Raw Data (Feedback Position)
Channel 2 Raw Data (INP Status)
Channel 3 Raw Data (Zero Velocity Index)
Length
PreTrgLen
Length

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.

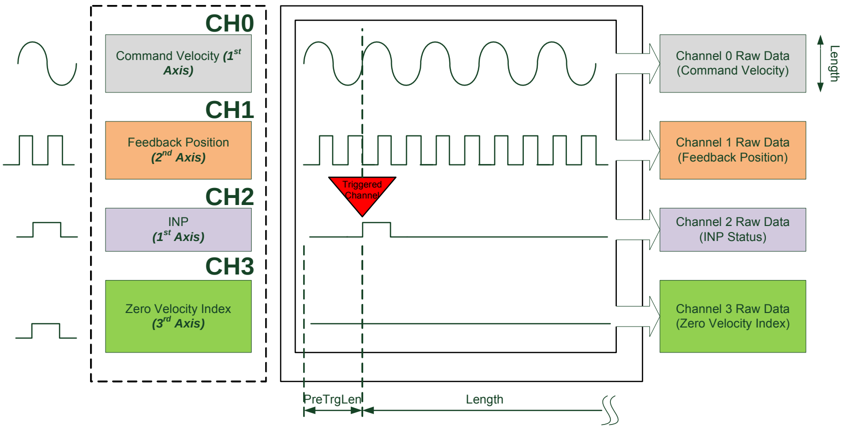
```
APS_set_sampling_param( Board_ID,
      SAMP_PA_SRC_CH0, SAMP_CMD_VEL ); //Set
      channel_0 sampling source
```

```
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 );
//…
```

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.


CH0
Command Velocity (1st Axis)
CH1
Feedback Position (2nd Axis)
CH2
INP (1st Axis)
CH3
Zero Velocity Index (3rd Axis)
Triggered Channel
Channel 0 Raw Data (Command Velocity)
Channel 1 Raw Data (Feedback Position)
Channel 2 Raw Data (INP Status)
Channel 3 Raw Data (Zero Velocity Index)
Length
PreTrgLen
Length

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
```

```
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 );

//…
```

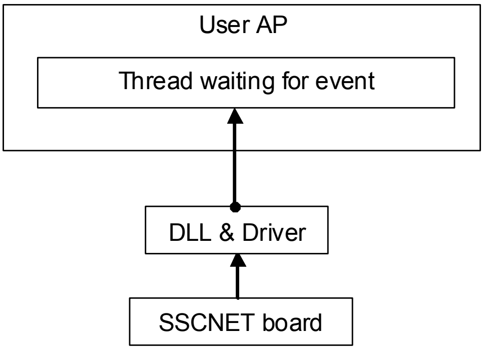
Finally, you need to define the storage length for sampling raw data. After the function APS_wait_trigger_sampling() was executed, the 1024 raw data was stored in the DataArr. Other advantage was also offered in this function, user is able to set the parameter ‘PreTrgLen’ and it defines the length of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

4.9 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 PCI-8253/6 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 PCI-8253/6 board, use the **APS_int_enable**() function. It acts as an **ON_OFF** switch. Once disabled, the PCI-8253/6 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 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 4-8: Interrupt Factor Table

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	--

4.10 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 motion starts.

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

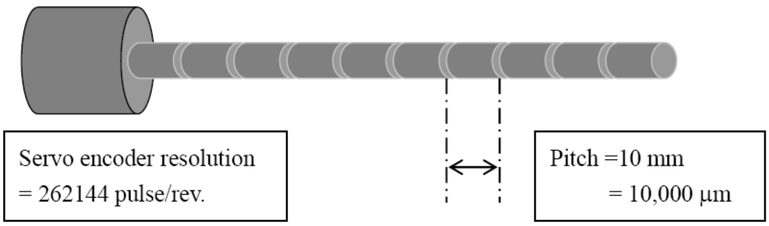
User-defines resolution = 2^N; N= 12 ~ 18

(Axis parameter: PRA_EGEAR)

For example, by using Mitsubishi MR-J3A 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)

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

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

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

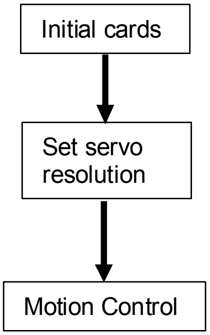
Distance: 1,000 pulse = 1,000 pulse X 1.221 μm/pulse = 1221 μm

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

= 1221 μm/sec.

Programming flow example:

Flow Corresponding functions



APS_initial()

APS_set_axis_param()

APS_set_servo_on()

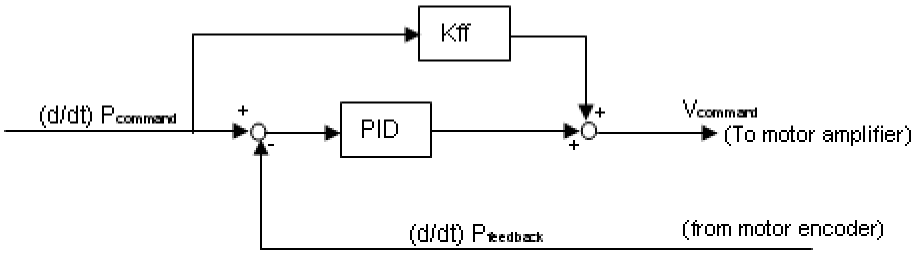
APS_velocity_move()

…

4.11 DSP-based Closed-loop Control

4.11.1 Closed-loop Control

In order to solve open loop gantry problem, the PCI-8253/6 can realize the gantry control with close-loop system. The control architecture is shown as below:



- ▶ $P_{command}$: Position command
- ▶ $P_{feedback}$: Position feedback
- ▶ d/dt: Differential operator
- ▶ Kff: Velocity feed forward gain (0 - 100%)
- ▶ PID: Controller gains contents Kp, Kd, Ki

The closed-loop control is relating to any system in which the command output was measured and compared to the feedback input. The command output is adjusted to reach the desired condition.

In motion control, the closed-loop control means a system utilizing an incremental encoder to generate correction signals in relation to desired parameters.

Servo systems are the most prevalent example of closed loop control. In a typical servo system, a motion operation is executed after the user issued a move command to the servo controller. The controller will calculate a velocity profile matching previously defined user trajectory which included the maximum velocity, acceleration, and deceleration parameters, etc. The controller applies a position/velocity command to the servo amplifier. Based on feedback from an incremental encoder the servo controller calculates the following error (difference between the actual position and the calculated position). The following error value is used to the PID filter to adjust the magnitude of the position / velocity command to the amplifier.

The significant advantage of a closed-loop system is that based on the constant corrections by the PID filter of the command voltage that based on the measured following error. The PID filter to properly respond to a given following error which the servo must be tuned. Tuning a servo parameter is a process in which the PID filter gain values are defined so that the response of the servo system to a given following error meets the requirements of the machine designer. Compared to an open loop system, which does not require PID filter tuning, the requirement of tuning a servo makes the setup of a closed-loop system a more complicated.

4.11.2 PID Filter Plus Feed Forward Gain

Proportional-integral-derivative, or PID, control is the most common type of controller used in industry. Depending on the specific application, any of the three components of PID can be implemented alone or in combination with another.

A proportional controller is the simplest and easiest to understand. In a typical system using proportional control, a desired setting or setpoint is compared to the output feedback signal. Proportional control, however, has limitations. A proportional controller cannot achieve a zero steady-state error. Increasing the gain in the system can decrease the steady-state error somewhat. However, when the gain is too high, the system overcorrects and may cause instability. Because a proportional controller cannot reduce the steady-state error to zero, it is used primarily in processes where gain can be made sufficiently large to reduce steady-state error, while at the same time maintain stability.

Proportional plus integral control (PI Controller)

In systems where the amount of error generated from a proportional controller cannot be tolerated, a technique is needed to respond to the steady-state error and drive it toward a smaller value, ideally zero. This technique is known as proportional plus integral, or PI, control.

A PI controller continuously adds up or integrates the error. An integrator by itself, with a constant input, would essentially provide an ever increasing output signal. However, used in conjunction with a proportional controller, as the error signal is integrated, the proportional control drives the error to a smaller value. It accomplishes the same effect as a large gain in a proportional controller without the adverse effect on stability. Thus, PI control is used in systems where the steady-state error must be reduced to smaller levels than with proportional control alone.

Because the integral section of the control operates within a specific reaction time, as the error signal approaches zero, the output from the proportional section goes to zero. However, because of the time constant, there could be a significant output from the integrator. This output causes the error to cross zero and create an error in the opposite direction. The error crossing zero produces overshoot which may cause instability. This tendency increases as the proportional gain increases and the integral reaction time or time constant is reduced.

A PI controller is used primarily in applications with frequent load disturbances and a minimum of setpoint changes. In practical applications using a PI controller, the proportional gain is set somewhat smaller than for a proportional-only control. The integral time constant is adjusted to provide either one overshoot or no overshoot.

Proportional plus derivative control (PD Controller)

Integral control is added to decrease long term or steady-state error. However, the faster and larger the process error, the greater the overshoot becomes, and the longer it takes to settle to an acceptable level. To deal with fast response times and large errors, derivative control is added.

Derivative control reduces the amount of time required for the output to return to the setpoint. It also reduces initial overshoot and has a stabilizing effect by damping the overshoot or oscillation. This permits the proportional gain to be set at higher values than with proportional-only control, and improves response time to system disturbances.

Derivative control is always used with proportional or PI control because it isn't capable of maintaining the error signal under steady-state control. It is used in applications that have sudden and relatively large disturbances.

PID controller

Each of the three above individual controllers has advantages. The proportional controller is the basic model. Increasing proportional gain to improve response produces instability. Integral control adds a complement which has a major effect on reducing long term errors. The derivative control has a major effect on transient disturbances when the proportional controller cannot respond fast enough.

The PID controller reduces steady-state error and responds rapidly to moves. Each has its own adjustment, and the various control adjustments interact.

Starting with the proportional control, gain is adjusted as high as possible to obtain adequate response to a change. When the integral component is added, the proportional gain must be reduced. If the integral reaction time constant is too fast (integral gain too large), it tends to become unstable. Adding the derivative component lets the proportional gain increase.

Feed forward Gain

In applications where the controller must compute and react quickly, it may happen that the controller cannot drive the error to zero fast enough to maintain precise position control. To overcome this, another technique called feed forward gain is used. This is a predictive type of control. In a typical feedback control system with an inner velocity and an outer position-control loop, velocity and acceleration feed forward is added after the position loop to the velocity loop. Predictive control depends upon earlier conditions, thus the signal has to bypass the position loop. In other words, the setpoint is compared to the next step in the process rather than the output. High-speed servo positioning applications, such as some robotics and X-Y-Z positioning systems, make use of this technique.

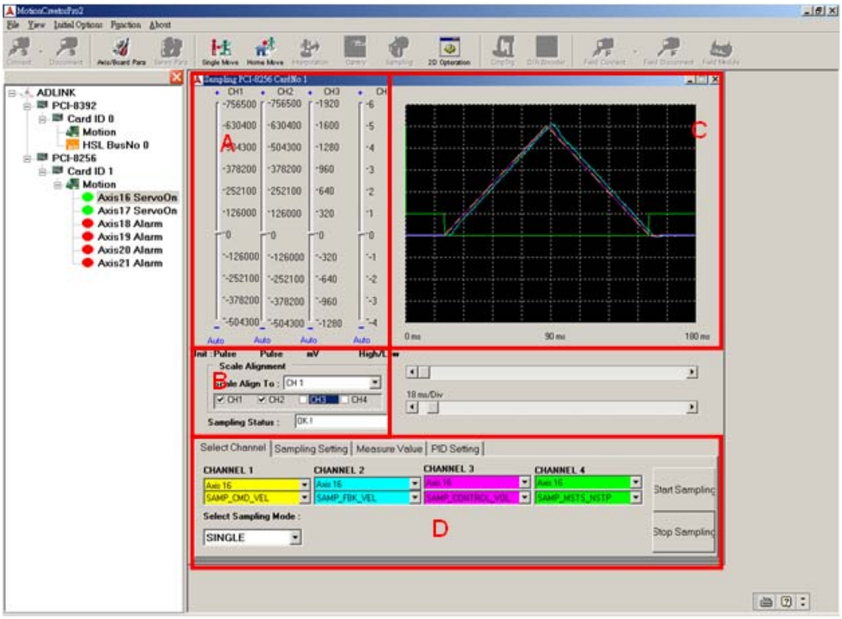
Tuning for an application

Tuning a controller refers to adjusting individual control parameters. This is first tested off-line with no load attached, then switched on-line with a load or the process running. The parameters of the PID are fine tuned to obtain the best response for the application. In a typical motion-control application with a velocity and position loop, the velocity-loop parameters must first be adjusted then the position-loop parameters.

If the gain is increased too much, the system becomes unstable and rings or oscillates. A motor typically produces an audible noise when the gain is set too high.

Sampling window for PID tuning and profile observation

The following illustrations present the PID tuning procedure and show the performance after adjusted PID parameters. ADLINK offers the sampling windows to analysis the motion profile instantly. By using the sampling window to observe the profile behavior after the parameters of PID and feed forward gain was tuned.



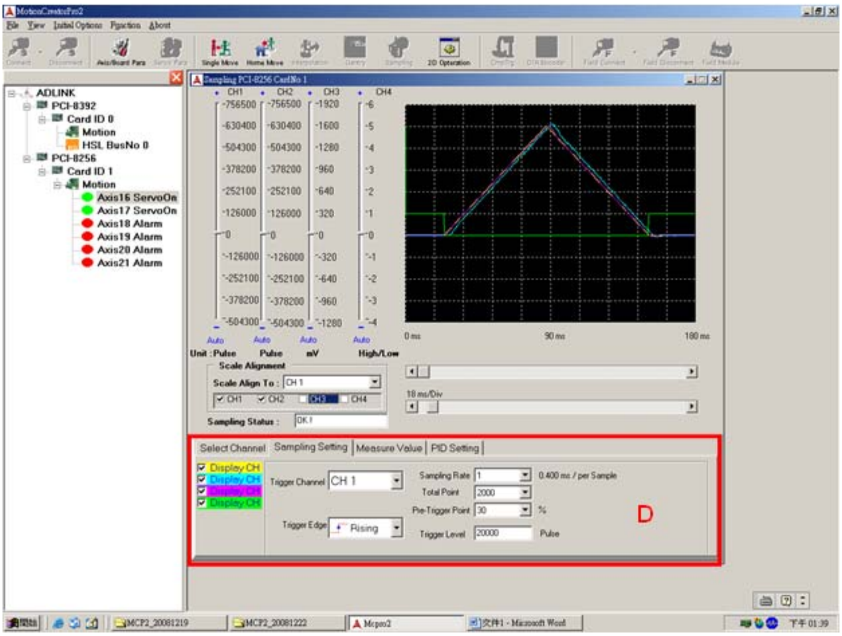
Operation instructions:

Block A: Vertical scale adjustment. The scale will be automatic adjusted when data is sampled. Users still can use the manual adjustment component to adjust the vertical scale and position. Users can also restore the automatic adjustment results by pressing Auto button. The unit of the channels will be displayed at bottom side too.

Block B: Scale alignment setting. Users can choose one channel which others will be aligned to. The aligned channels can be selected by check box of channels.

Block C: Sampling scope. Right click mouse button brings up measurement cursor. Users can use zoom in/out slider bar and shift slider bar to adjust the horizontal position and scale of the graph.

Block D, **Tab1**: Channel selections. There are four channels in the scope. Each channel can be chosen from any axes. Each channel's signal source can be chosen from the pull-down menu. The sampling mode can be selected as single trigger or repeat trigger mode. The start and stop sampling buttons are in this area too.

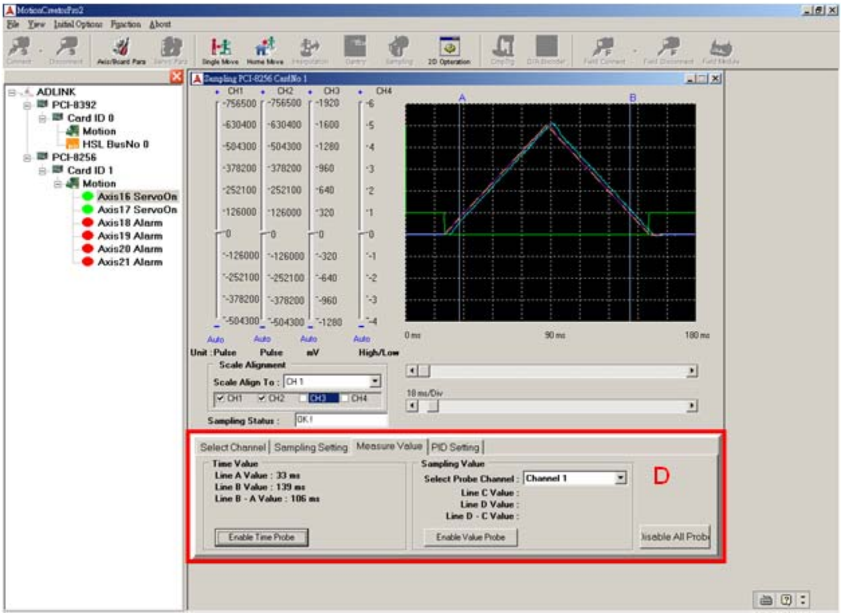


Scale Alignment
Scale Align To : CH 1
☑ CH1 ☑ CH2 ☐ CH3 ☐ CH4
Sampling Status : OK !

☑ Display CH
☑ Display CH
☑ Display CH
☑ Display CH

Trigger Channel	CH 1	Sampling Rate	1	0.400 ms / per Sample
		Total Point	2000	
		Pre-Trigger Point	30	%
Trigger Edge	Rising	Trigger Level	20000	Pulse

Block D, **Tab2**: Trigger condition for sampling. Users can select active channels and trigger source channels. The trigger conditions are selected from rising or falling edge of the signal. In this area, users can also set the sampling rate, total sampled points, pre-triggered percentage and trigger level. The unit of trigger level will changed automatically by selected trigger channel.

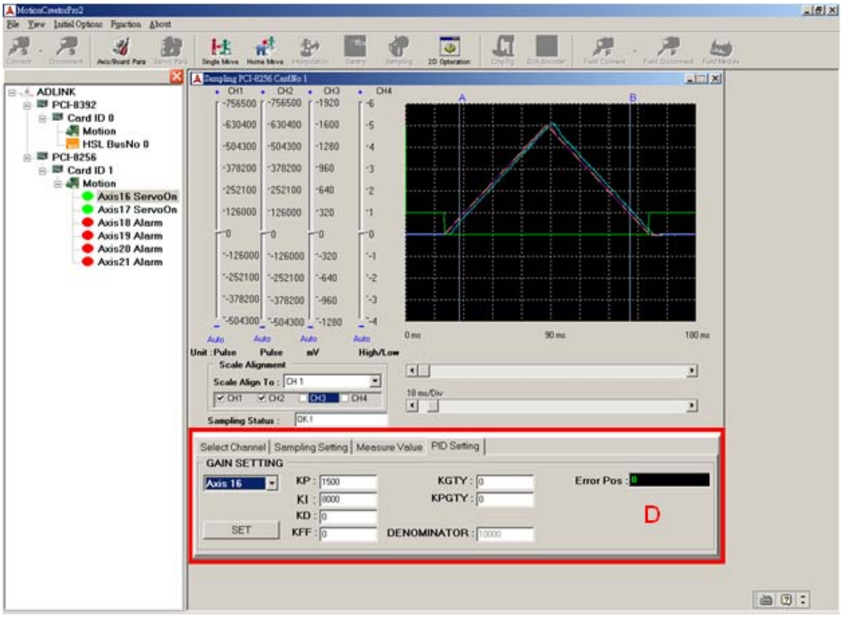


Scale Alignment
Scale Align To : CH 1
✓ CH1 ✓ CH2 CH3 CH4
Sampling Status : OK!

Time Value
Line A Value : 33 ms
Line B Value : 139 ms
Line B - A Value : 106 ms

Sampling Value
Select Probe Channel : Channel 1
Line C Value :
Line D Value :
Line D - C Value :

Block D, Tab3: Measuring cursor area. There are time probe cursor in vertical and value probe cursor in horizontal. It will display each cursor’s time or value and their subtraction value.



Select Channel | Sampling Setting | Measure Value | PID Setting

GAIN SETTING

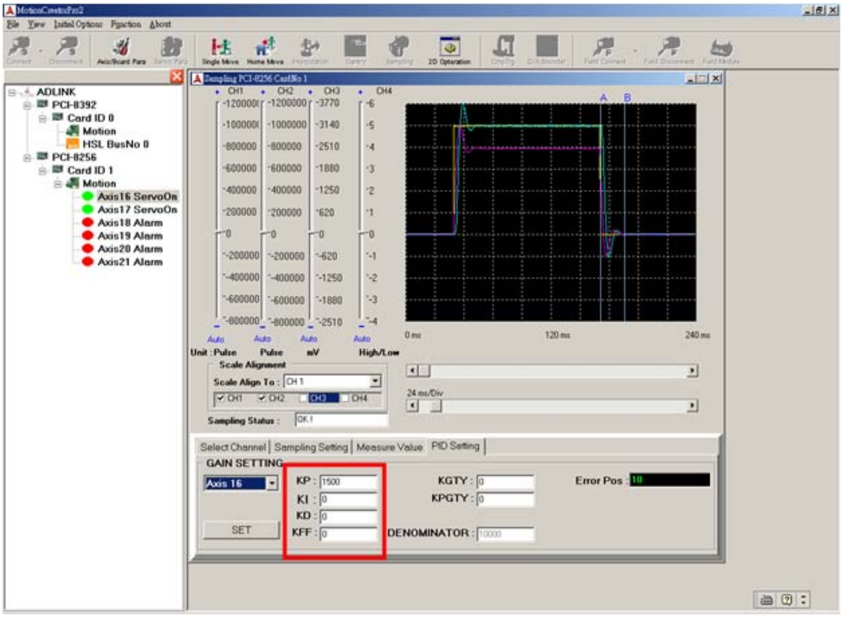
Axis 16

KP : 1500
KI : 8000
KD : 0
KFF : 0
KGTY : 0
KPGTY : 0
DENOMINATOR : 10000
Error Pos : 0

SET

Block D, Tab4: PID setting area. Users can select one axis to tune the dynamic response. There are P, I, D and FF gain for one axis. If using gantry mode, there are two additional gains, KGTY and KGPTY for cross-coupling tuning. In this page, it also displays current error position for steady state reference and the parameter's denominator. It means if KP=1500 and demoninator=10000, that means the actual KP in the control loop is 0.15.

Tuning by Step Response

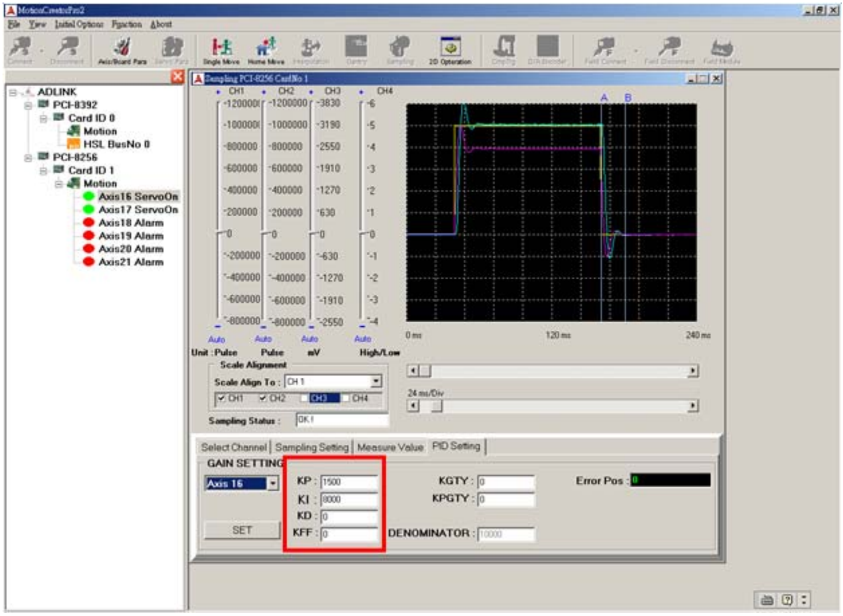


Axis 16

KP : 1500
KI : 0
KD : 0
KFF : 0
KGTY : 0
KPGTY : 0
DENOMINATOR : 10000
Error Pos : 10

Operation Instructions:

1. Create a step input speed profile by setting VS,VM,VE at the same value. Once they are at the same value, the motion kernel will ignore acceleration and deceleration time.
2. First, set a KP value to watch its response. Normally, we will increase the KP gradually until we see the oscillation response.
3. In the above example, we set KP to 1500 and see the oscillation response. You can see that the error position is not zero due to DC gain or other factor.



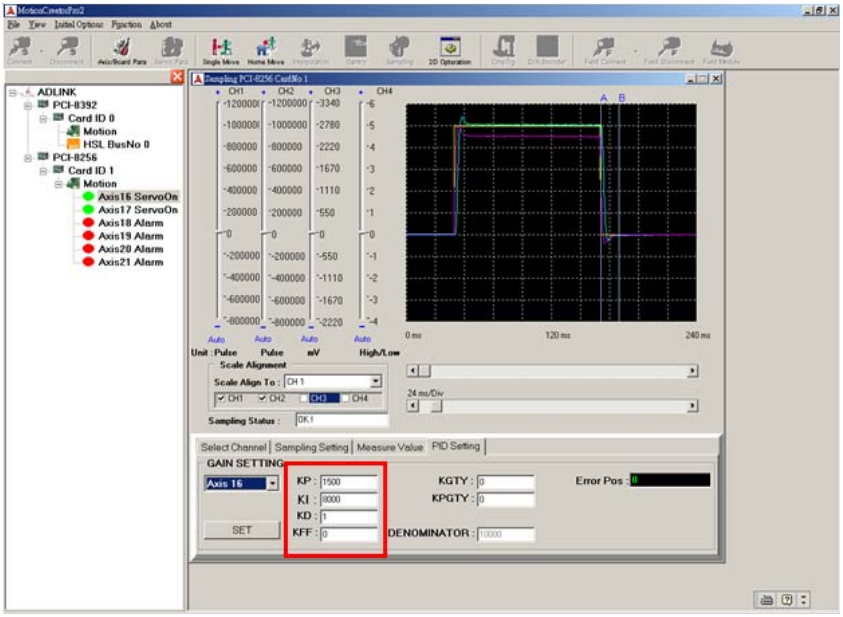
Scale Alignment
Scale Align To : CH 1
Sampling Status : OK!

Axis 16

KP : 1500
KI : 8000
KD : 0
KFF : 0
KGTY : 0
KPGTY : 0
DENOMINATOR : 10000
Error Pos : 0

Operation Instructions:

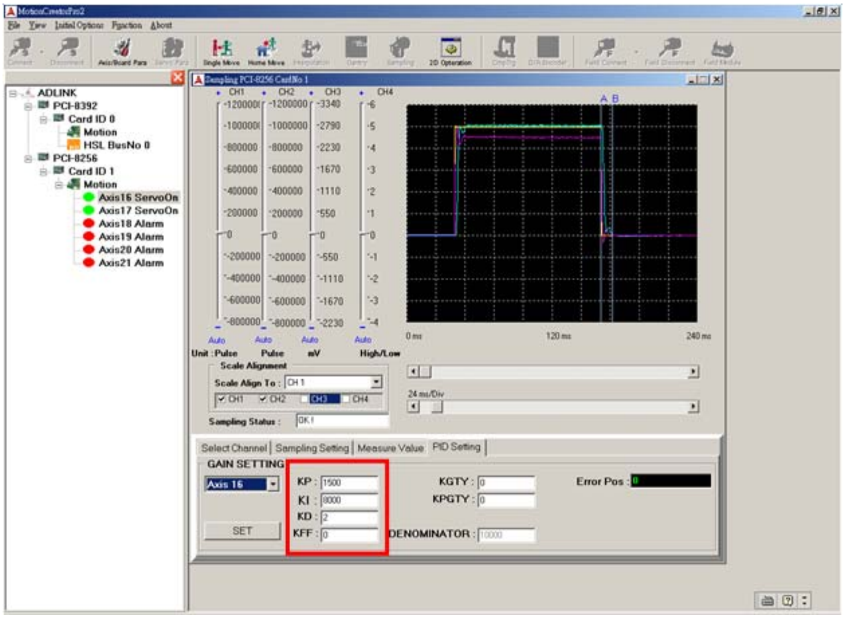
4. Try to add KI gain to eliminate DC gain which results in steady state error.
5. We set a value 8000 and the position error converges in an acceptable time.
6. But the response still has overshoot.



KP : 1500
KI : 8000
KD : 1
KFF : 0
KGTY : 0
KPGTY : 0
DENOMINATOR : 10000

Operation Instructions:

7. Try to suppress the overshoot by adding KD gain.
8. We add 1 in KD gain and see it does suppress the over-shoot.
9. It becomes a less under-damped system and the steady state error remains kept at 0

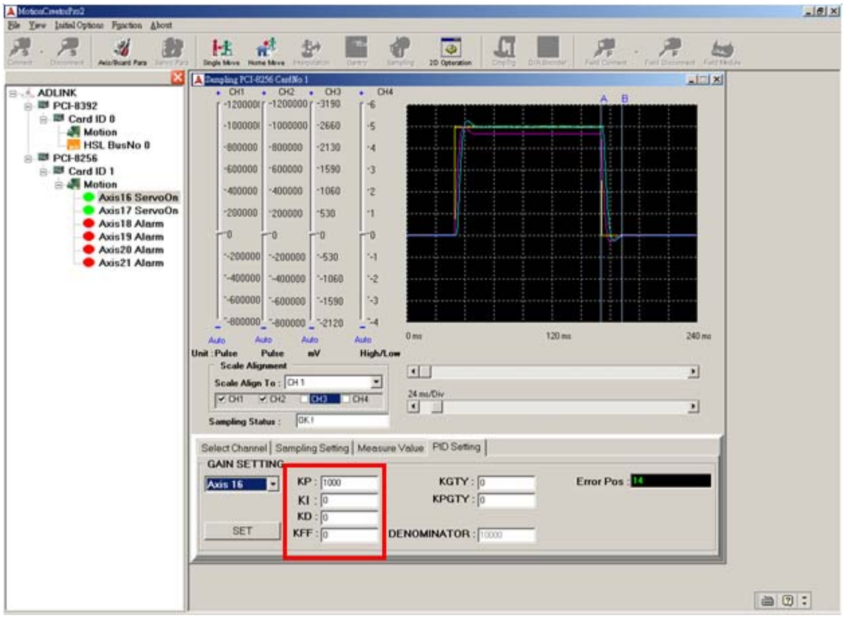


Scale Alignment
Scale Align To : CH 1
☑ CH1 ☑ CH2 ☐ CH3 ☐ CH4
Sampling Status : OK!

GAIN SETTING
Axis 16
KP : 1500
KI : 8000
KD : 2
KFF : 0
KGTY : 0
KPGTY : 0
DENOMINATOR : 10000
Error Pos : 0

Operation Instructions:

10. Try to increase the KD value to 2 and it becomes an over-damped system



Scale Align To : CH 1

✓ CH1 ✓ CH2 CH3 CH4

Sampling Status : OK !

GAIN SETTING

Axis 16

KP : 1000

KI : 0

KD : 0

KFF : 0

KGTY : 0

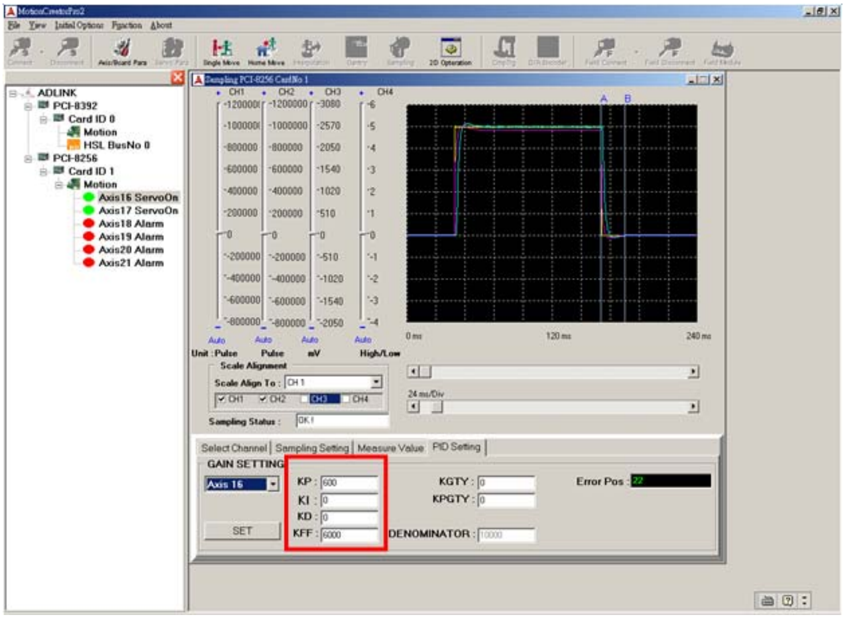
KPGTY : 0

DENOMINATOR : 10000

Error Pos : 14

Operation Instructions:

11. Try the other method to tune the system. We will use Kff this time. It can increase the response time and reduce following error.
12. Set a smaller KP value to 1000 than previous method.
13. You can see the overshoot is smaller than before but the following error is bigger.

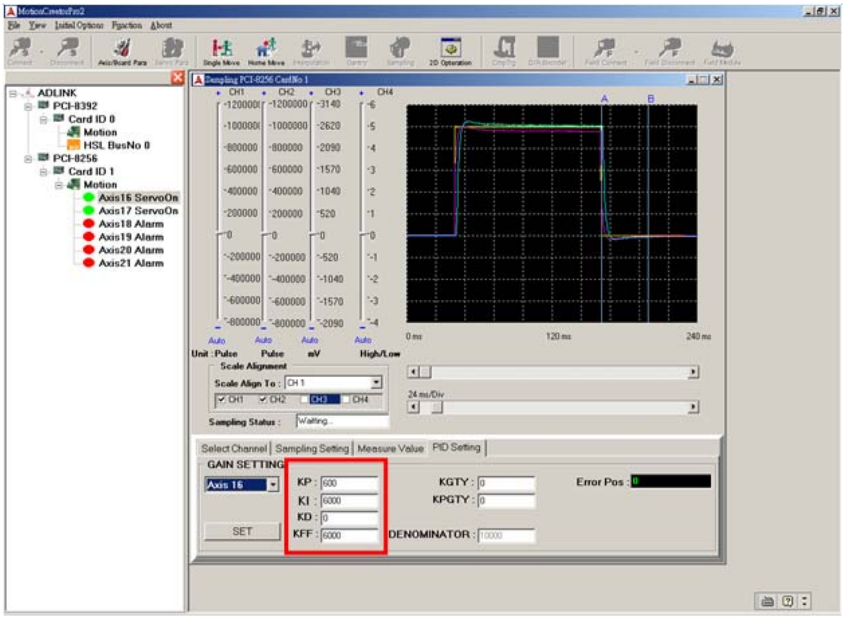


Sampling Status : OK!

KP : 600	KGTY : 0	Error Pos : 22
KI : 0	KPGTY : 0	
KD : 0		
KFF : 6000	DENOMINATOR : 10000	

Operation Instructions:

14. Try to add Kff to 6000 and reduce KP. We can see the overshoot is smaller and following error is smaller too.



Scale Align To : CH 1

Sampling Status : Waiting...

Axis 16

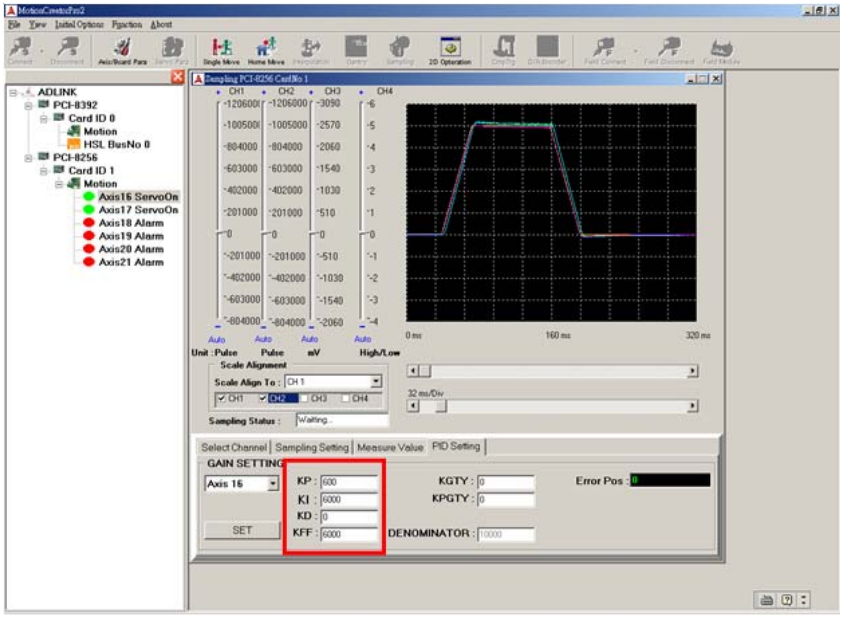
KP : 600
KI : 6000
KD : 0
KFF : 6000

KGTY : 0
KPGTY : 0
DENOMINATOR : 10000

Error Pos : 0

Operation Instructions:

15. Now, try to eliminate the steady state error by adding KI gain to 6000.
16. You can see the error is 0 but the stable time is longer. That's a trade-off.
17. Users can try more combinations to fit their key requirements



Unit : Pulse Pulse mV High/Low

Scale Alignment

Scale Align To : CH 1

☑ CH1 ☑ CH2 ☐ CH3 ☐ CH4

Sampling Status : Waiting..

GAIN SETTING

Axis 16

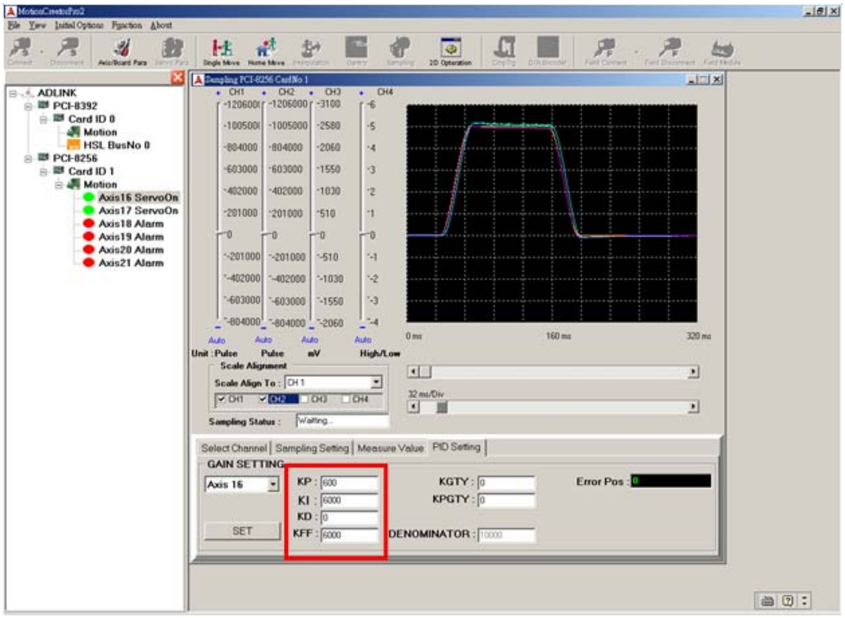
KP : 600
KI : 6000
KD : 0
KFF : 6000

KGTY : 0
KPGTY : 0
DENOMINATOR : 10000

Error Pos : 0

Operation Instructions:

18. If we use the step response tuning results on a trapezoidal profile, we can get a good response too.



ADLINK
PCI-8392
Card ID 0
Motion
HSL BusNo 0
PCI-8256
Card ID 1
Motion
Axis16 ServoOn
Axis17 ServoOn
Axis18 Alarm
Axis19 Alarm
Axis20 Alarm
Axis21 Alarm

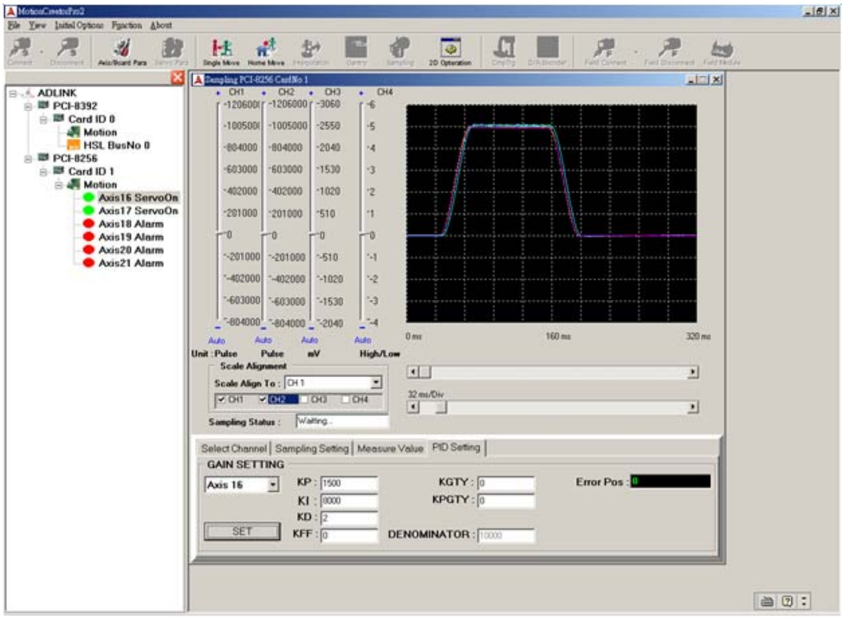
Unit : Pulse Pulse mV High/Low

Scale Alignment
Scale Align To : CH 1
CH1 ✓ CH2 ✓ CH3 CH4
Sampling Status : Waiting...

GAIN SETTING
Axis 16
KP : 600
KI : 6000
KD : 0
KFF : 6000
KGTY : 0
KPGTY : 0
DENOMINATOR : 10000
Error Pos : 0

Operation Instructions:

19. If we use the step response tuning results on a S-curve profile, we can get a good response too.



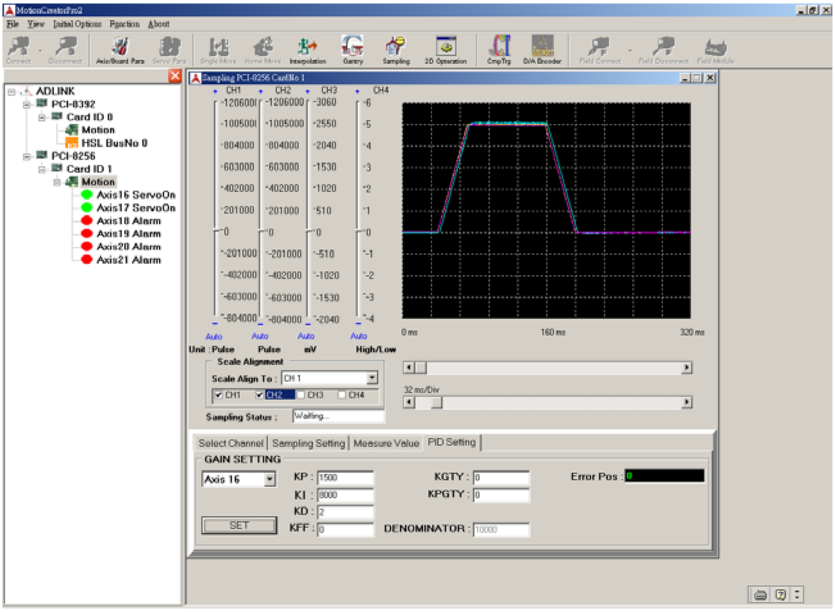
ADLINK
PCI-8392
Card ID 0
Motion
HSL BusNo 0
PCI-8256
Card ID 1
Motion
● Axis16 ServoOn
● Axis17 ServoOn
● Axis18 Alarm
● Axis19 Alarm
● Axis20 Alarm
● Axis21 Alarm

Scale Alignment
Scale Align To : CH 1
[✓] CH1 [✓] CH2 [ ] CH3 [ ] CH4
Sampling Status : Waiting..

GAIN SETTING
Axis 16
KP : 1500
KI : 8000
KD : 2
KFF : 0
KGTY : 0
KPGTY : 0
DENOMINATOR : 10000
Error Pos : 0

Operation Instructions:

20. We can also use PID step response tuning results for trapezoidal profile and S-curve profile.


MotionCreatorPro2
File View Initial Options Function About
Connect Disconnect Axis/Board Para Servo Para Single Move Home Move Interpolation Gantry Sampling 2D Operation CmpTrg D/A Encoder Field Connect Field Disconnect Field Module
ADLINK
PCI-8392
Card ID 0
Motion
HSL BusNo 0
PCI-8256
Card ID 1
Motion
Axis16 ServoOn
Axis17 ServoOn
Axis18 Alarm
Axis19 Alarm
Axis20 Alarm
Axis21 Alarm
Sampling PCI-8256 CardNo 1
CH1 CH2 CH3 CH4
0 ms 160 ms 320 ms
32 ms/Div
Select Channel Sampling Setting Measure Value PID Setting
GAIN SETTING
SET

Unit : Pulse Pulse mV High/Low

Scale Alignment

Scale Align To : CH 1

✓ CH1 ✓ CH2 CH3 CH4

Sampling Status : Waiting...

Axis 16

KP : 1500

KI : 8000

KD : 2

KFF : 0

KGTY : 0

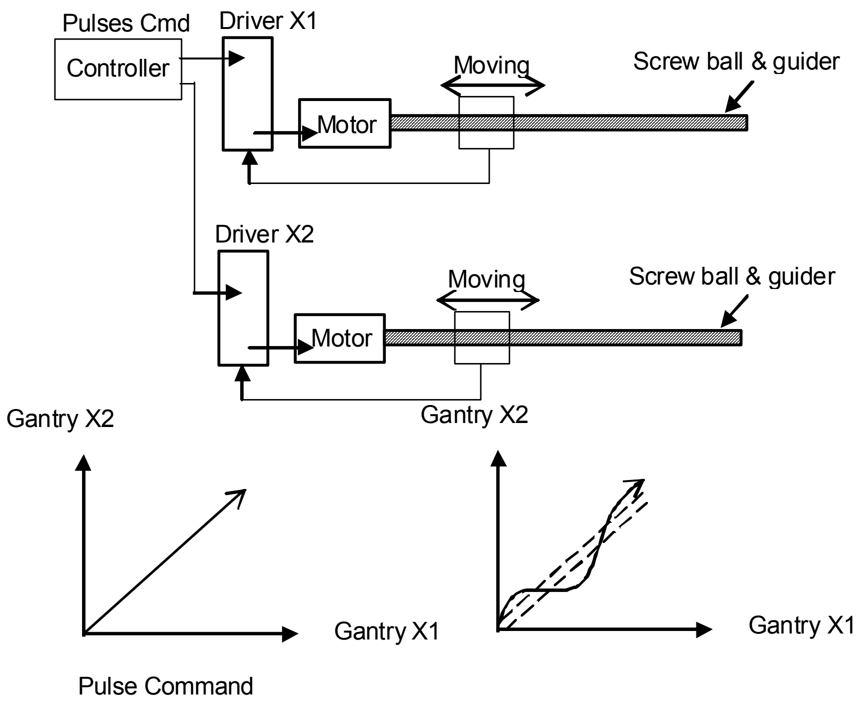
KPGTY : 0

DENOMINATOR : 10000

Error Pos : 0

4.11.3 Gantry Mode

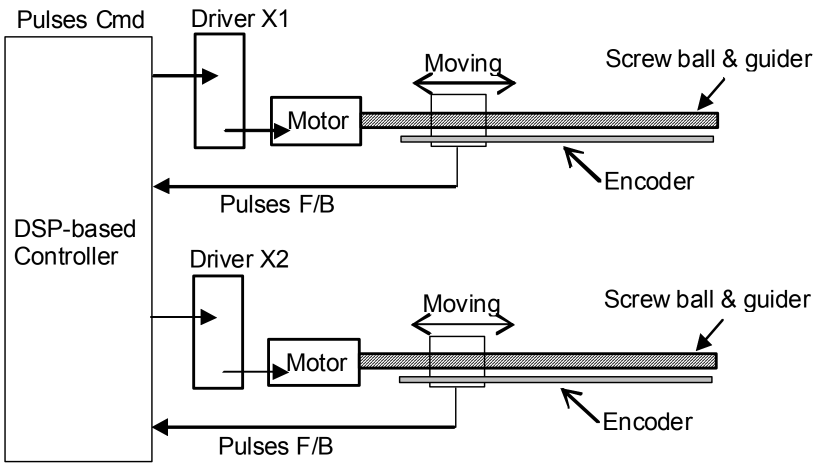
Gantry function is defind from mechanical structure. Normally we have two motors controlling two parallel linear stages. These two motors must move similutaneously. The motion controller must control these two motors individually but synchronously. It looks like 1:1 electric gearing in master-slave motion 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:

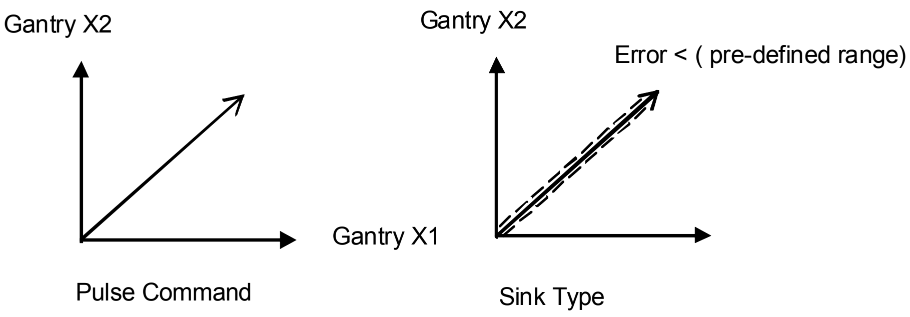


If the gantry X1 and X2 has different loading and servo parameters, these two axes will not move simultaneously.

4.11.4 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:

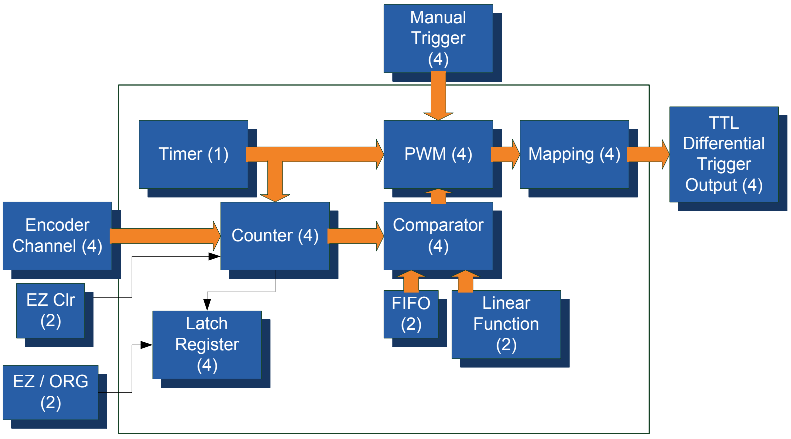




4.12 Position Compared and Trigger pulse ouput

4.12.1 Architecture

The PCI-8253/6 provides 2/4 trigger output channels where 1/2 of them is applied to high-speed position compared function and up to 1 MHz trigger pulse output. Other 1/2 channels are applied to low-speed application and support pulse output frequency up to 25 KHz at least. Inside the PCI-8253/6, all the position comparison and mapping tasks will be executed within the FPGA. The function block figures as following:



4.12.2 Encoder Channels

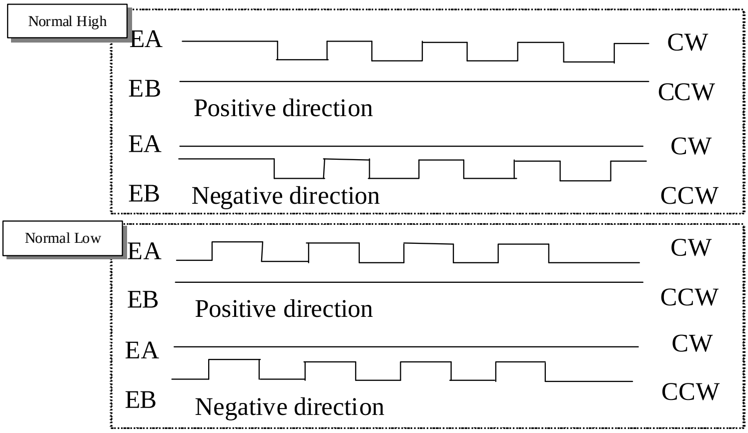
The FIFO will automatically load the comparing points into 32-bit comparator herein. User is able to set the comparing source as four encoder inputs and two timer channels. Due to mapping logic setting, the several combinations are provides for user to perform continuously trigger output function. The comparator will compare the data from counter and FIFO individually. Besides, the comparator is also able to compare the counter and comparing data which was produced by embedded Linear function. At very short time, user can retain and retrieve the current counter data by general purposed digital input. Linear function will automatically load the ext comparing points with fixed incremental value into 32-bit comparator. Like FIFO usage purpose, the new comparing points will be loaded once the previous comparing point is compared and pulse is triggered.

The PCI-8253/6 has a 32-bit binary up/down counter managing the present position feedback for whole dual channel.

It accepts 2 types of pulse inputs: (1). Dual pulse mode (CW/CCW) (2) 90° phase shifted signals (AB phase mode). 90° phase shifted signals maybe multiplied by a factor of 1, 2 or 4. 4x AB phase mode is the most commonly used in incremental encoder inputs. For example, if a rotary encoder has 2000 pulses per phase (A or B phase), then the value read from the counter will be 8000 pulses per turn or –8000 pulses per turn depending on its rotating direction. The three options will be explained as follows.

Dual Pulse Mode (CW/CCW Mode)

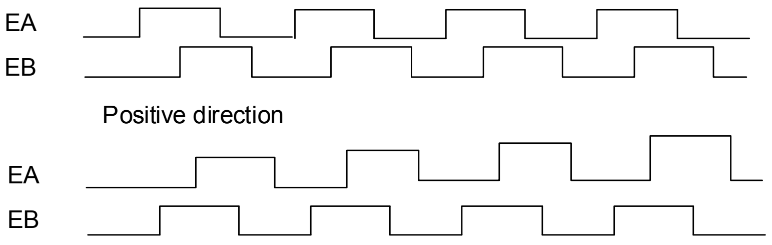
In this mode, EA is dedicated to count the pulses from external source and view it as clockwise direction (CW). EB is dedicated to count the pulses from external source and view it as counterclock-wise direction (CCW). Simply put, EA counts up and EB counts down. User can decide the normal high or normal low for those two channels according to users' devices. The following diagrams show the normal high and normal low cases individually.



90° phase shifted signals (AB phase Mode)

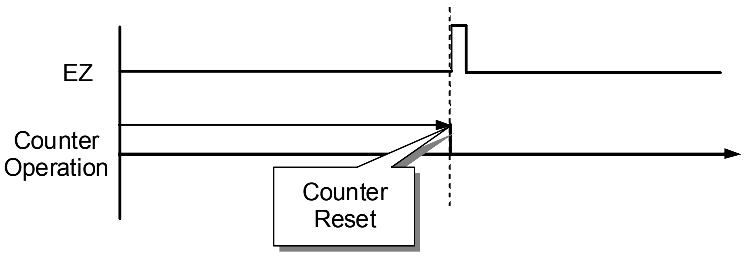
In this mode, EA signal is a 90° phase leading or lagging to EB signal. “Lead” or “lag” of phase difference between two signals is caused by the turning direction of the motor. The up/down counters counts up when the phase of EA signal leads the phase of EB signal.

The following diagram shows the waveform.



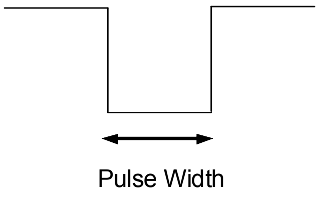
4.12.3 Index Input (EZ)

The PCI-8253/6 can clear the counter value as zero according to the edge of EZ signal. Homing by edge can let users meet best homing positioning purpose. Rising or falling edge is supported. The following diagram shows the case about the homing by rising edge.



4.12.4 Trigger Pulse Width

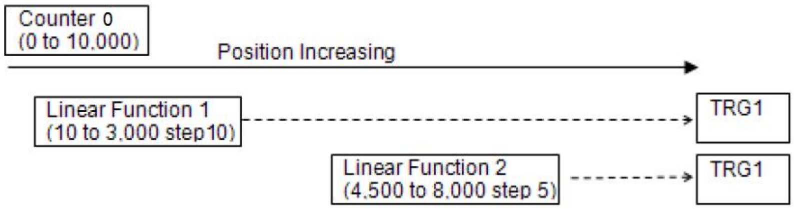
For different applications, the trigger pulse width requirement is different. As for this reason, the trigger pulse width can be adjustable. The available values are from 0.2us to 6.55ms. The maximum frequency is up to 1 MHz.



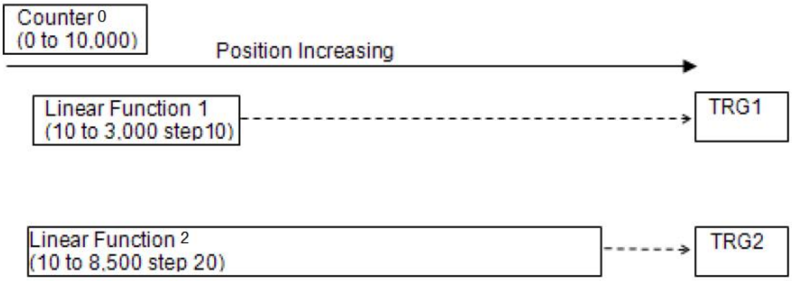
4.12.5 Linear Function

Linear function is used to generate a new comparing position point by a fixed incremental value linear function, D=D’ + A. D means a calculated comparing position, the linear data. D’ means a previous comparing position. Every time the position was compared, hence a new data is calculated by adding ‘A’, the fixed incremental value. User can implement a continuous triggering function promptly by using linear function trigger.

There are 2 linear functions in PCI-8253/6 which supports to each PWM channel individually and each function is independent. It allows that linear function range overlapped is possible when system operating. Each linear function has its own comparator and the comparator could be linked to any one of two counters. By this features, users can produce many kinds of trigger modes. Please see the following diagrams:



Take two linear functions. Set the trigger interval and range as shown in the diagram. Set these two linear functions to counter0 and also comparator0. Set two linear functions to same trigger output pin TRG1. After these settings, when the counters start counting from 0 to 10,000, the trigger pins will output pulses respectively when the compare conditions are met inside linear function. From TRG1 pins, the trigger pulse output will be observed which was generated by linear function 1 and linear function 2 sequentially.



For other case, when the counters start counting from 0 to 10,000, the trigger pins will output pulses respectively when the compare conditions are met inside linear function. This diagram is also able to present the overlap status of TRG1 output if makes TRG 2 instead of TRG1.

4.12.6 FIFO Function

FIFO is first-in-first-out storage. It is used for storing some preset position data for comparing. Every time the position is compared, a new data is retrieve from FIFO into comparator. This mechanism makes a the continuous triggering function.

Continuous triggering is fulfilled by linear function and FIFO. These two modes have their own comparators and can be used at the same time. The FIFO mode is usually used on random comparing data condition. Users can preset these data into FIFO and perform continuous triggering. In PCI-8253/6, there are two FIFO and have the capacity with 15 comparing data space. The FIFO support to anyone of four PWM channels.

4.12.7 PWM & Mapping function

PWM is used for adjusting pulse width of trigger. It could also be switched to a toggle mode. In this mode, the pulse level will change from low to high or high to low at every time when compared.

Mapping means eight trigger signals are not one-to-one mapping to three comparators. Trigger output channels are able to , For example, Comparator 1 could be linked to trigger channel 2. Comparator 2 could be linked to trigger channel 1 and 4. Comparator 2 and 4 could be linked to channel 3.

4.12.8 Position Comparison

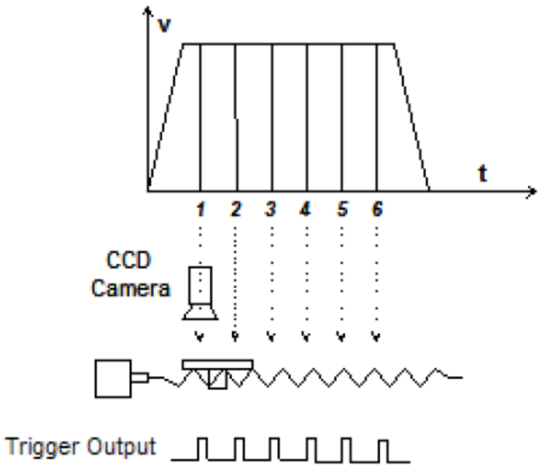
The PCI-8253/6 provides position comparison functions for 2/4 channels. Once the counter reaches a preset value set by the user, the PCI-8253/6 will generate the trigger pulse. TRGx pins are for trigger pulse output channels which were designed by TTL differential output type. Therefore, the differential signal is able to reduce noise-effect efficiently while trigger pulse signal transmitting.

The comparing method is “equal”. Consequently, when the counter value is exactly equal to the pre-set value by users, the trigger pulse will be generated.

At the same time, the next comparing points saved in FIFO or linear function will automatically loaded into comparator. The following is an example for continuous trigger application.

Example: Using the continuous position comparison function.

In this application, the table is controlled by the motion command, and the CCD Camera is controlled by the position comparison output of the PCI-8253/6. An image of the moving object is easily obtained.



4.12.9 Position Latch

The position latch function is fulfilled by EZ/ORG signal. Once the EZ/ORG signal is active, the counter value of its latch channel will be saved to latched counter value at the same time. User can read the latch register any time.

4.12.10Timer Function

There is a timers on PCI-8253/6. The timer can be set to counters to simulate encoder inputs. It can also output to trigger pins directly. The timer is designed by a down-counter. Users must set a counter value into timer for down counting. Once the timer counter reaches zero, the timer will output a pulse to trigger pin or increase encoder counter by 1. The down counting speed is 100ns and the maximum counter value is 30-bit. User can set the timer interval from 100ns to 26ms.

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 8253/6 cards.

Note that MotionCreatorPro2 is only available for Windows 2000/XP with a screen resolution higher than 1024x768. Recommended screen resolution is 1024x768. It cannot be executed under the DOS environment.

5.1 Execute MotionCreatorPro 2

After installing the software drivers for the card in Windows 2000/XP, the MotionCreatorPro 2 program can be located at <chosen path >\PCI-Motion\MotionCreatorPro2. To execute the program, double click on the executable file or use **Start** -> **Program Files** -> **ADLINK** -> **PCI-8253/6** -> **MotionCreatorPro 2**.

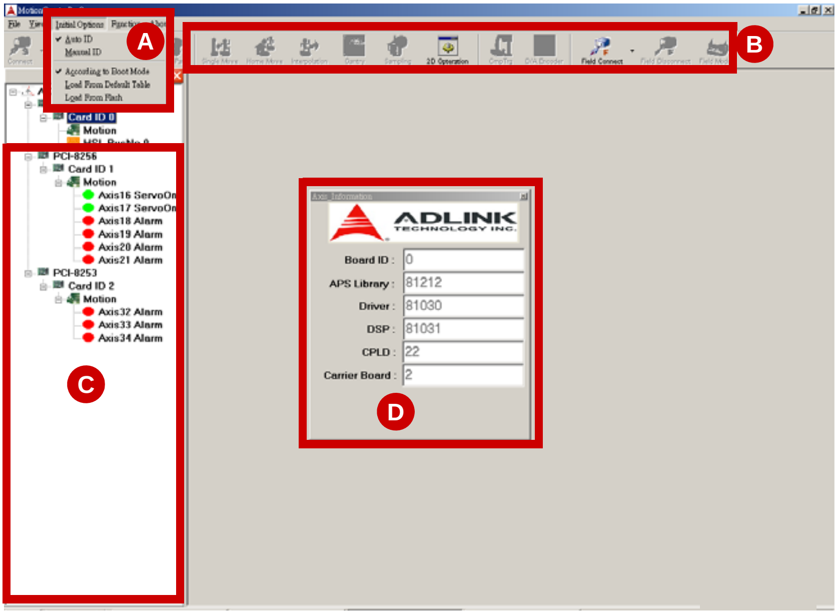
5.2 About MotionCreatorPro 2

Before running MotionCreatorPro 2 please note that MotionCreatorPro 2 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.

5.3 MotionCreatorPro 2 Forms

5.3.1 Main Menu

The main menu appears after running MotionCreatorPro 2. Refer to the following illustrations for a description of the available functions:



Board ID :	0
APS Library :	81212
Driver :	81030
DSP :	81031
CPLD :	22
Carrier Board :	2

A. Initial options for manual or auto ID and axis parameter loading mode. All the settings will be active the next time MotionCreatorPro 2 runs.

B. Icons for operation modes. Some will be active when a motion item in the tree view is selected and some will be active when an axis item is selected.

Button Functions: 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. **** PCI-8392/8392H only***
	SSCNET Disconnect	Disconnect axis. **** PCI-8392/8392H only***
	Axis/Board Configuration	Configure axis or board parameter.
	SSCNET Servo Parameter Configuration	Configure parameter of MITSUBINSHI J3 Servo Driver. **** PCI-8392/8392H only***

- Movement

Button	Function	Description
	Single Movement	Single Axis movement (PTP), includes absolute and relative functions.
	Home Return Movement	Home return movement.
	Interpolation	Interpolation function.
	Sampling	Sampling function, it can select source and drew its profile.
	2D Movement	Execute 2D motion movement

▶ HSL

Button	Function	Description
	HSL Connect	Connect HSL module. Please select the baud rate at the right side of the button and connect. **** PCI-8392 only***
	HSL Disconnect	Disconnect HSL module. **** PCI-8392 only***
	HSL Module Test	If connected correctly, select the module and use this to perform a module test. **** PCI-8392 only***

▶ Others

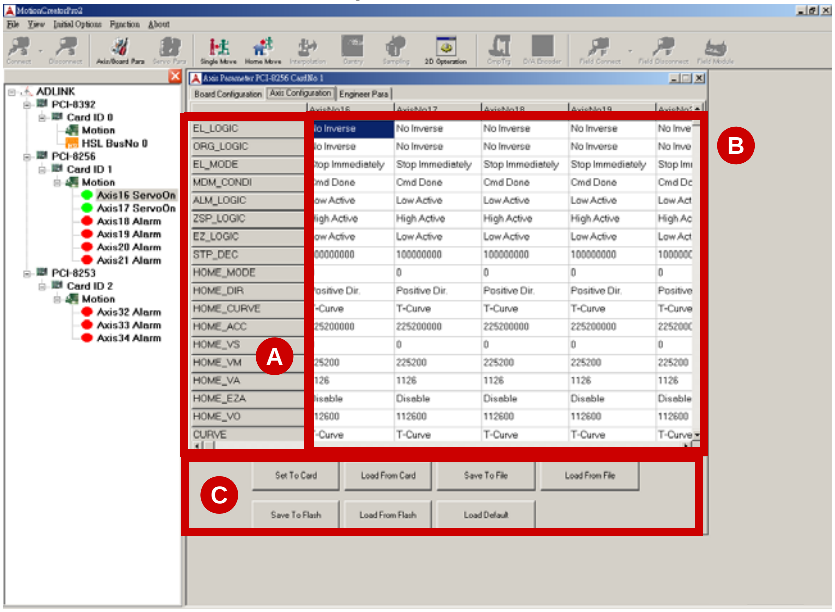
Button	Function	Description
	Gantry Motion	Execute gantry motion manipulation and tune the cross-coupling gain value.
	Position Compare and Trigger Output	Fill the compared position and select comparison methods.
	Monitor I/O Status	Includes DI/O, AI/O, and encoder counter values

C. Displays all motion boards found by MotionCreatorPro 2. The tree view can display both motion axes and field bus I/O.

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

D. Board information (software, firmware, and hardware versions).

5.3.2 Parameter Management



	AxisNo16	AxisNo17	AxisNo18	AxisNo19	AxisNo
EL_LOGIC	lo Inverse	No Inverse	No Inverse	No Inverse	No Inve
ORG_LOGIC	lo Inverse	No Inverse	No Inverse	No Inverse	No Inve
EL_MODE	top Immediately	Stop Immediately	Stop Immediately	Stop Immediately	Stop Imı
MDM_CONDI	md Done	Cmd Done	Cmd Done	Cmd Done	Cmd Dc
ALM_LOGIC	ow Active	Low Active	Low Active	Low Active	Low Act
ZSP_LOGIC	ligh Active	High Active	High Active	High Active	High Ac
EZ_LOGIC	ow Active	Low Active	Low Active	Low Active	Low Act
STP_DEC	00000000	100000000	100000000	100000000	100000C
HOME_MODE		0	0	0	0
HOME_DIR	ositive Dir.	Positive Dir.	Positive Dir.	Positive Dir.	Positive
HOME_CURVE	-Curve	T-Curve	T-Curve	T-Curve	T-Curve
HOME_ACC	25200000	225200000	225200000	225200000	225200C
HOME_VS		0	0	0	0
HOME_VM	25200	225200	225200	225200	225200
HOME_VA	126	1126	1126	1126	1126
HOME_EZA	isable	Disable	Disable	Disable	Disable
HOME_VO	12600	112600	112600	112600	112600
CURVE	-Curve	T-Curve	T-Curve	T-Curve	T-Curve

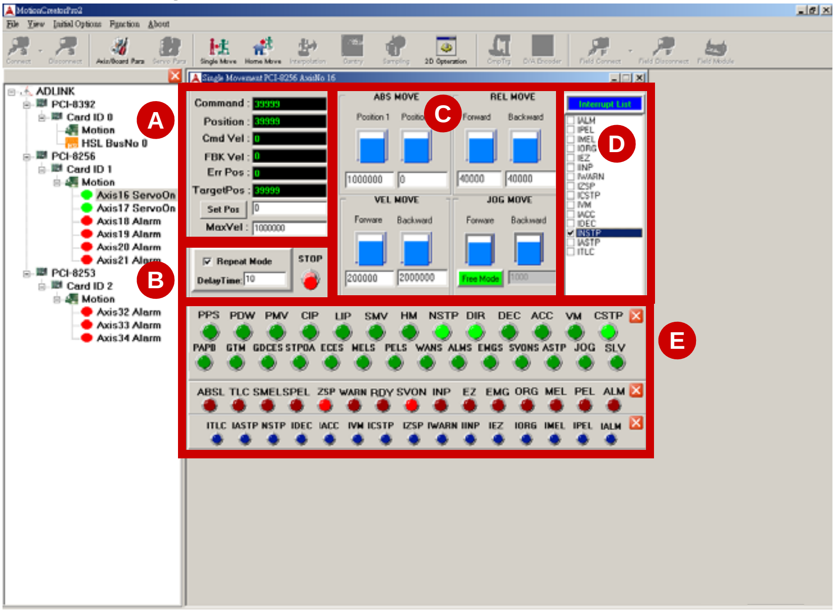
A. Parameter type display

B. Parameter values of all axes

C. Parameter management button to load/save parameters. Set to card must be used to activate this table.

Hint: Apply all parameters to other axes with the right mouse button.

5.3.3 Single Movement


MotionCreatorPro2
File View Initial Options Function About
Connect Disconnect Axis/Board Para Servo Para Single Move Home Move Interpolation Gantry Sampling 2D Operation CmpTrg DIA Encoder Field Connect Field Disconnect Field Module
ADLINK
PCI-8392
Card ID 0
Motion
HSL BusNo 0
PCI-8256
Card ID 1
Motion
Axis16 ServoOn
Axis17 ServoOn
Axis18 Alarm
Axis19 Alarm
Axis20 Alarm
Axis21 Alarm
PCI-8253
Card ID 2
Motion
Axis32 Alarm
Axis33 Alarm
Axis34 Alarm
Single Movement PCI-8256 AxisNo 16
A
B
C
D
E
ABS MOVE
REL MOVE
VEL MOVE
JOG MOVE
STOP
Interrupt List
IALM IPEL IMEL IORG IEZ IINP IWARN IZSP ICSTP IVM IACC IDEC INSTP IASTP ITLC
PPS PDW PMV CIP LIP SMV HM NSTP DIR DEC ACC VM CSTP
PAPB GTM GDCES STPOA ECES MELS PELS WANS ALMS EMGS SVONS ASTP JOG SLV
ABSL TLC SMEL SPEL ZSP WARN RDY SVON INP EZ EMG ORG MEL PEL ALM
ITLC IASTP NSTP IDEC IACC IVM ICSTP IZSP IWARN IINP IEZ IORG IMEL IPEL IALM

Command : 39999
Position : 39999
Cmd Vel : 0
FBK Vel : 0
Err Pos : 0
TargetPos : 39999
Set Pos 0
MaxVel : 1000000

☑ Repeat Mode
DelayTime: 10

ABS MOVE — Position 1: 1000000, Position 2: 0
REL MOVE — Forward: 40000, Backward: 40000
VEL MOVE — Forware: 200000, Backward: 2000000
JOG MOVE — Free Mode, 1000

Interrupt List: ☑ INSTP

A. Command, feedback, error and target position information. Command and feedback speed information. The minimum speed value may limit by speed calculation cycle time for low speed display.

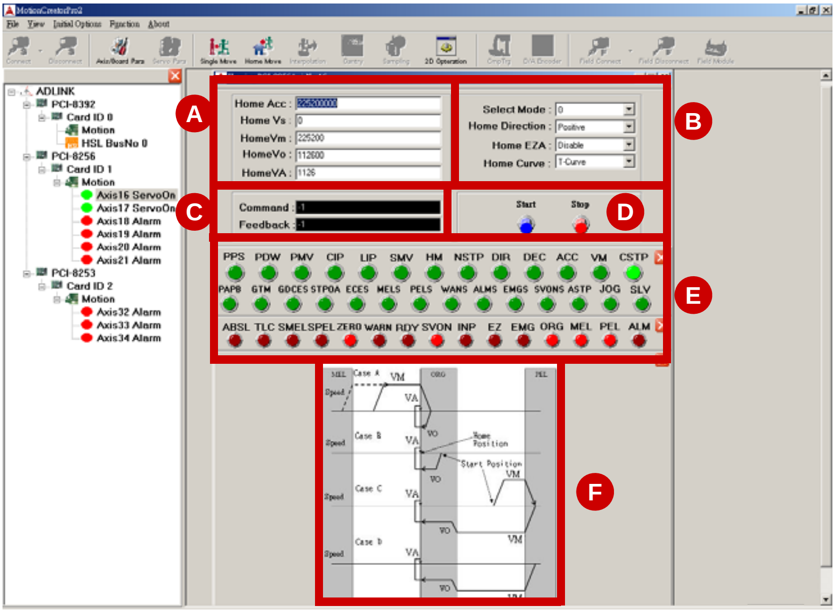
B. Optional operation setting and button. The repeat mode check box can be used in relative and absolute mode. The axes will move between two positions or forward/backward distance cyclically. You can set the delay time between each move in unit of mini-second. The minimum value is 1 ms. The stop button is for relative, absolute and velocity modes.

C. Operation buttons and setting for 4 modes. You can switch operation between relative, absolute and velocity modes. The parameters of each mode must be set before operation, such as position 1 & 2, forward/backward distance and forward/backward velocity. Remember to set MaxVel before executing relative and absolute mode. When using jog mode, the other three modes will be disabled. There are two options for job modes. One is free and the other is step. The Jog parameters are set from axis parameter table. The step offset in step mode can be set in this area too.

D. Interrupt factor setting area

E. Motion status, I/O status and interrupt status display area.

5.3.4 Home Return



Home Acc : 225200000
Home Vs : 0
HomeVm : 225200
HomeVo : 112600
HomeVA : 1126

Select Mode : 0
Home Direction : Positive
Home EZA : Disable
Home Curve : T-Curve

Command : -1
Feedback : -1

A. Speed parameters of the homing profile. Please refer to the figure in “F”.

B. Mode settings of homing functions. Select one of the items inform the pull-down menu

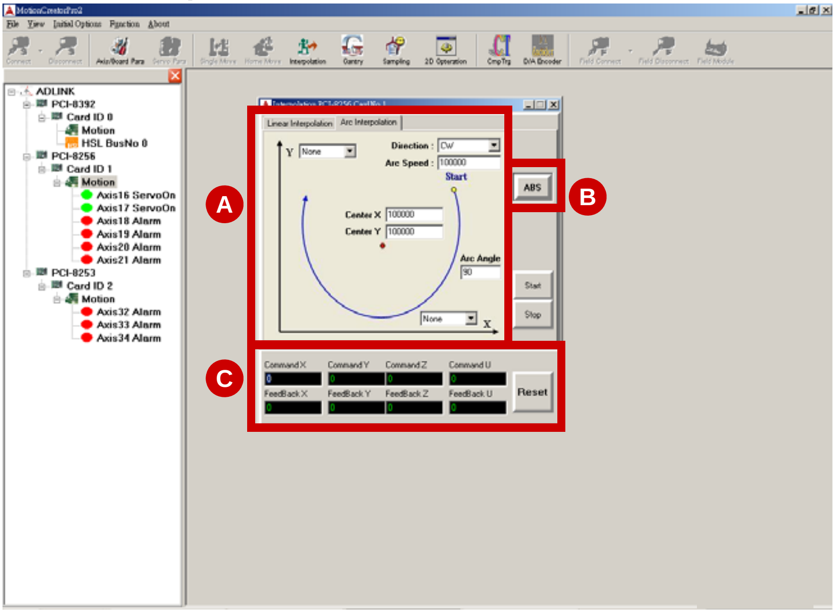
C. Command and position information when homing. After homing completes, the command counter will be reset to zero at the edge of ORG ON when VA speed.

D. Operation button to start homing or stop/abort homing functions.

E. Motion and I/O status when homing

F. The timing chart of the homing function.

5.3.5 Interpolation

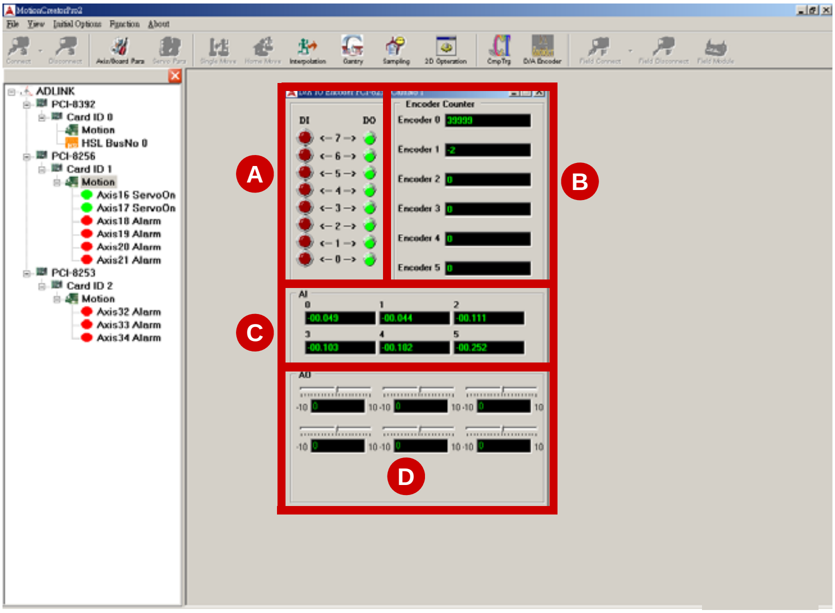


A. Interpolation axes selection and operation parameter settings, including center position in ARC mode or target position in Linear mode. The arc angle parameter can be larger than 360.

B. Absolute or relative interpolation mode selection. In ARC mode, it is about center position. In Linear mode, it is about target position.

C. Command and position information. In Arc mode, only two of them will be active.

5.3.6 General I/O Status



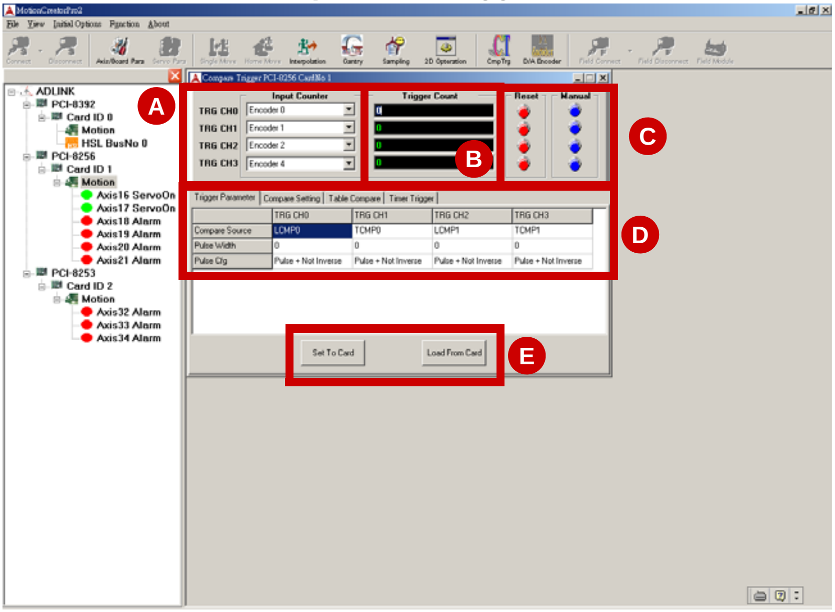
A. Digital I/O operation area. For the PCI-8253, there are only 0-3 bits available for operations.

B. Encoder counter display area in units of pulses

C. Analog input display area in units of voltage

D. Analog output operation area. You can drag the bar to make analog output. The output value will be displayed too.

5.3.7 Position Compare and Trigger Functions



	Input Counter
TRG CH0	Encoder 0
TRG CH1	Encoder 1
TRG CH2	Encoder 2
TRG CH3	Encoder 4

	TRG CH0	TRG CH1	TRG CH2	TRG CH3
Compare Source	LCMP0	TCMP0	LCMP1	TCMP1
Pulse Width	0	0	0	0
Pulse Cfg	Pulse + Not Inverse	Pulse + Not Inverse	Pulse + Not Inverse	Pulse + Not Inverse

A. Input counter selection for each trigger pin

B. The total triggered counts

C. Reset: reset triggered counts value. Manual: manual trigger output when button is pressed.

D. Trigger parameter table settings

E. After the parameters are modified in the table above, press the set to card button to activate them. You can press load from card to above table too.

Hint: Any operation mode such as single axes for multi-axis operations can be used for input counter counting for testing trigger counts.