PCI-8134/PCI-8134A

4-Axis Servo / Stepper
Motion Control Card
User's Guide

Manual Rev.: 3.00
Revision Date: Sept 7, 2012
Part No: 50-11173-1000

Recycled Paper

Getting service from ADLINK

♦Customer Satisfaction is always the most important thing for ADLINK Tech Inc. If you need any help or service, please contact us and get it.

ADLINK Technology Inc.			
Web Site	http://www.adlinktech.com		
Sales & Service	service@adlinktech.com		
Technical Support	NuDAQ	nudaq@adlinktech.com	
	Automation	automation@adlinktech.com	
	NuIPC	nuipc@adlinktech.com	
	NuPRO/EBC	nupro@adlinktech.com	
TEL	+886-2-82265877	FAX	+886-2-82265717
Address	9F, No. 166, Jian Yi Road, Chungho City, Taipei, 235 Taiwan		

♦Please inform or FAX us of your detailed information for a prompt, satisfactory and constant service.

Detailed Company Information			
Company/Organization			
Contact Person			
E-mail Address			
Address			
Country			
TEL		FAX	
Web Site			
Questions			
Product Model			
Environment to Use	☐OS _____ ☐Computer Brand _____ ☐M/B: ☐CPU: ☐Chipset: ☐Bios: ☐Video Card: ☐Network Interface Card: ☐Other:		
Challenge Description			

Suggestions for ADLINK	

Table of Contents

About This Guide

The PCI-8134 was EOL in May, 2011. ADLINK offers the new PCI-8134A as a line replacement. While most PCI-8134A functions are fully compatible with legacy PCI-8134 functions, certain differences require changes in application, as outlined in this document.

Chapter1, "Introduction", gives an overview of the product features, applications, and specifications.

Chapter2, "Installation", describes how to install the PCI-8134/PCI-8134A.

Chapter3, "Signal Connection", describes the connectors' pin assignment and how to connect the outside signal and devices with the PCI-8134/PCI-8134A.

Chapter4, "Operation Theorem", describes detail operations of the PCI-8134/PCI-8134A.

Chapter5, “Motion Creator & Motion Creator Pro”, describe how to utilize a Microsoft Windows based utility program to configure and test running the PCI-8134/PCI-8134A.

Chapter6, "C/C++ Function Library", describes high-level programming interface in C/C++ language. It helps programmer to control PCI-8134/PCI-8134A in high level language style.

Chapter7, "Another Function Library (8134A.lib) ", describes high-level programming interface. It helps programmer to control PCI-
8134 in high level language style.

Chapter8, “Connection Example” shows some typical connection examples between PCI-8134/PCI-8134A and servo driver and stepping driver.

1

Introduction

The PCI-8134/PCI-8134A is a 4-axis motion control card with PCI interface. It can generate high frequency pulses to drive stepping motors and servo motors. Multiple PCI-8134/PCI-8134A cards can be used in one system. Incremental encoder interface on all four axes provide the ability to correct for positioning errors generated by inaccurate mechanical transmissions. In addition, mechanical sensor interface, servo motor interface and general purpose I/O signals are provided for system integration.

Figure 1.1 shows the function block diagram of PCI-8134/PCI-8134A card. PCI-8134/PCI-8134A uses motion ASIC to perform 4-axis motion control. These ASICs are incorporate Nippon Pulse Motor. The motion control functions include linear and S-curve acceleration/deceleration, interpolation between two axes, continuous motion, in positioning and home return are done by the ASIC. Since these functions needing complex computations are done internally on the ASIC, the PC's CPU is free to supervise and perform other tasks.

Motion Creator a Microsoft Windows-based application included with the PCI-8134/PCI-8134A card for supporting application development. Motion Creator is very helpful for debugging a motion control system during the design phase of a project. The on-screen monitor shows all installed axis information and I/O signals status of PCI-8134/PCI-8134A cards. In addition to Motion Creator, both DOS and Windows version function library are included for programmers using C++ and Visual Basic language. Several sample programs are given to illustrate how to use the function library.

The following flowcharts show recommending processes for using this manual to develop an application. Please also refer to the relative chapters for details of each step.

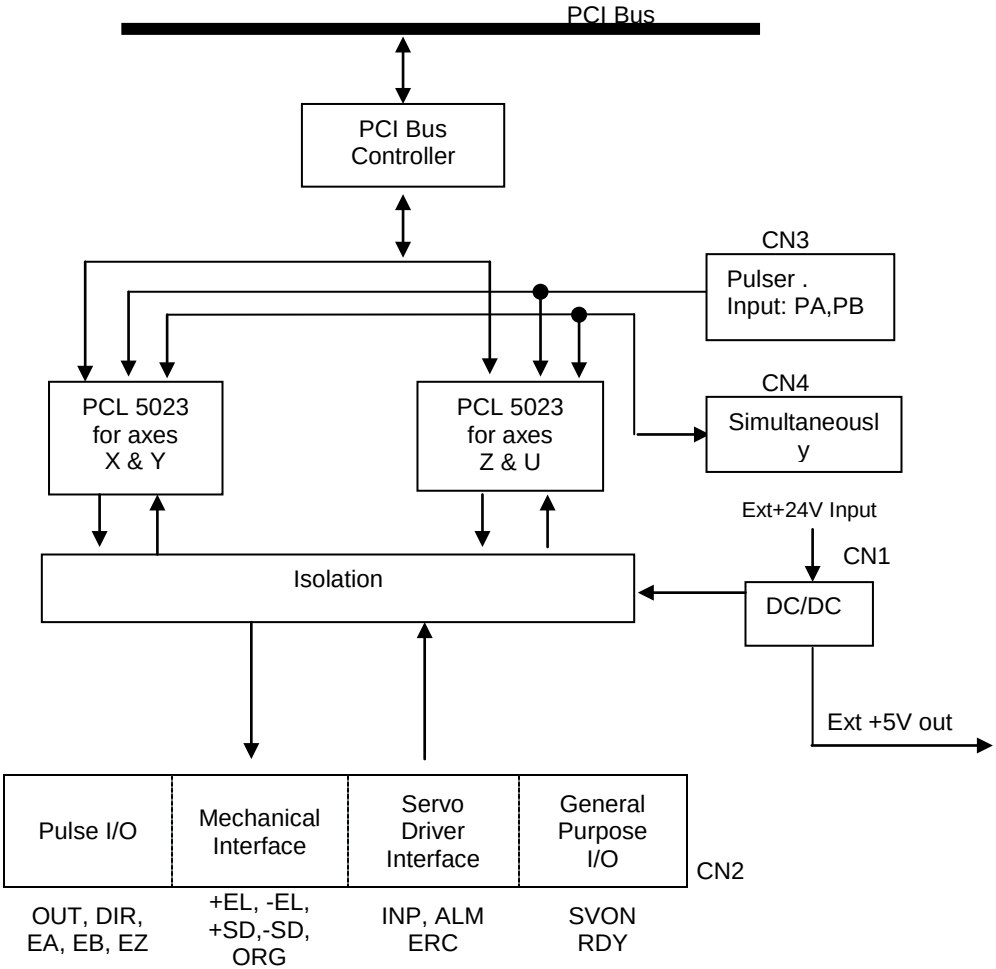


Figure 1.1 Block Diagram of PCI-8134

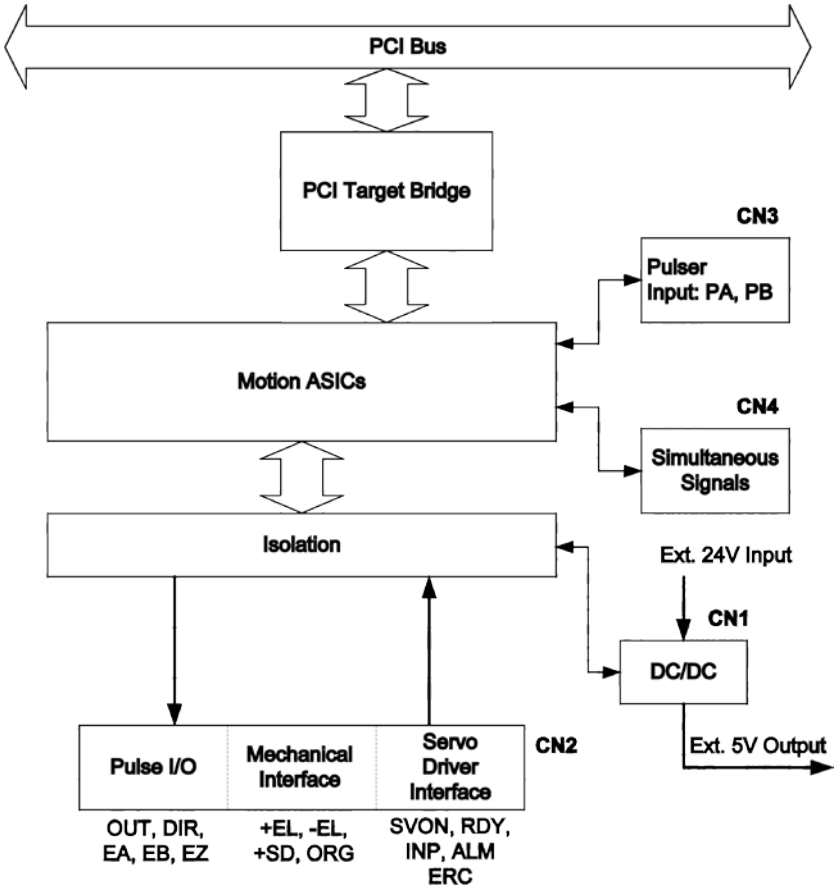


Figure 1.2 Block Diagram of PCI-8134A

1.1 Features

The following lists summarize the main features of the PCI-8134 motion control system.

- 32-bit PCI-Bus, plug and play.
- 4 axes of step and direction pulse output for controlling stepping or servomotor.
- Maximum output frequency of 2.4 Mpps
- Pulse output options: OUT/DIR, CE/CCW
- Pulse input options: CW/CCW, AB phase x1, x2, x4
- 2-axis linear interpolation.
- 28-bit up/down counter for incremental encoder feedback.
- Home switch, index signal, positive and negative limit switches interface provided for all axes
- Trapezoidal and S-curve velocity profiles for all modes
- Programmable interrupt sources
- Change Speed on the Fly.
- Simultaneous start/stop motion on multiple axes.
- Manual pulser input interface.
- Software supports maximum up to 12 PCI-8134/PCI-8134A cards (48 axes) operation.
- Compact, half size PCB.
- Motion Creator Microsoft Windows based application development software.

1.2 Specifications

◆ **Applicable Motors:**

- Stepping motors.
- AC or DC servomotors with pulse train input servo-drives.

◆ **Performance:**

- Number of controllable axes: 4
- Maximum pulse output frequency: 2.4Mpps, linear, trapezoidal or S-Curve velocity profile drive.
- Position pulse setting range: 0~268,435,455 pulses (28-bit).
- Ramping-down point setting range: 0 to 16777215
- Acceleration / deceleration rate setting range: 1 to 65535(16bit)
- Up / down counter counting range: 0~268,435,455 (28-bit.) or –134,217,728 to +134,217,727

- Pulse rate setting steps: 0 to 2.4Mpps.

◆ **I/O Signals:**

- Input/Output Signals for each axis
- All I/O signal are optically isolated with 2500Vrms isolation voltage
- Command pulse output pins: OUT and DIR.
- Incremental encoder signals input pins: EA and EB.
- Encoder index signal input pin: EZ.
- Mechanical limit/switch signal input pins: ±EL, SD and ORG.
- Servomotor interface I/O pins: INP, ALM and ERC.
- General purpose digital output pin: SVON.
- General purpose digital input pin: RDY.
- Pulser signal input pin: PA and PB.
- Simultaneous Start/Stop signal I/O pins: STA and STP.

◆ **General Specifications**

- Connectors: 100-pin SCSI-type connector
- Operating Temperature: 0° C ~ 50° C
- Storage Temperature: -20° C ~ 80° C
- Humidity: 5 ~ 85%, non-condensing
- Power Consumption:

* Slot power supply (input): +5V DC ±5%, 900mA max.
* External power supply (input): +24V DC ±5%, 500mA max.
* External power supply (output): +5V DC ±5%, 500mA, max.
* PCI-8134 Dimensions: 164mm(L) X 98.4mm(W)
* PCI-8134A Dimensions: 185mm(L) X 100mm(W)

1.3 Software Support

1.3.1 Programming Library

Windows® XP/7 DLLs are provided for the PCI-8134 and PCI-8134A. These function libraries are shipped with the board.

1.3.2 Motion Creator

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

1.4 Compatible Terminal Boards

ADLINK provides servos & steppers with terminal boards for easy connection, specifically boards DIN-814M0, DIN-814M-J3A0, DIN-814Y0, DIN-814P-A40 for connection to dedicated servo drives. Steppers or other servo brands can be connected with general purpose terminal boards DIN-814-GP and DIN-100S0. Compatible servos are as follows.

Servo	Terminal Board
Mitsubishi J2 Super	DIN-814M0
Mitsubishi J3A	DIN-814M-J3A0
Yaskawa Sigma II	DIN-814Y0
Panasonic MINAS A4	DIN-814P-A40
Other Serovs and Steppers	DIN-814-GP (specific for cable selection) DIN-100S0

2

Installation

This chapter describes how to install the PCI-8134/PCI-8134A, according to the following procedure.

- Check Package Contents (Section 2.1)
- Check the PCB (Section 2.2)
- Install the hardware (Section 2.3)
- Install the software driver (Section 2.4)
- Acquaint yourself with the I/O signal connections (Chapter 3) and their operation (Chapter 4)
- Check the connector pin assignments and wiring

2.1 Package Contents

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

- PCI-8134/PCI-8134A 4-Axis Servo / Stepper Motion Control Card
- ADLINK All-in-one Compact Disc
- User's Guide Manual

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

2.2 PCI-8134/PCI-8134A Outline Drawing

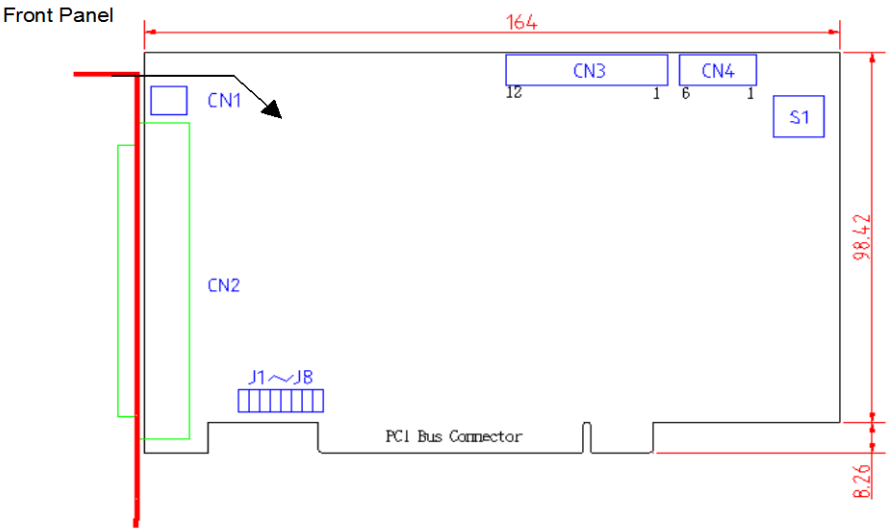


Figure 2.1 PCB Layout of the PCI-8134

CN1: External Power Input Connector
CN2: Input / Output Signal Connector
CN3: Manual Pulser Signal Connector
CN4: Simultaneous Start / Stop Connector

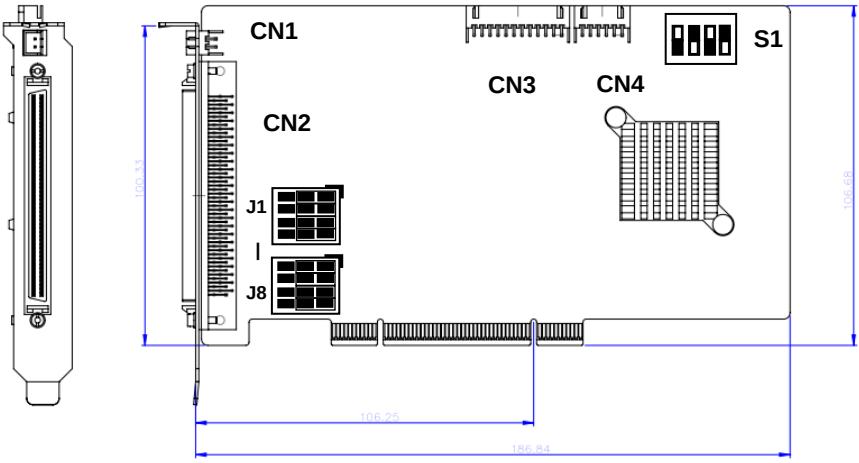


Figure 2.2 PCB Layout of the PCI-8134A

CN1: External Power Input Connector
CN2: Input / Output Signal Connector
CN3: Manual Pulser Signal Connector
CN4: Simultaneous Start / Stop Connector
J1-J8: Pulse output type selection
S1: Polarity of end-limited switch selection

2.3 Hardware Installation

2.3.1 Hardware configuration

The PCI-8134/PCI-8134A has a plug and play PCI controller on board. The memory usage (I/O port locations) of the PCI card is assigned by 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 will probably have both PCI and ISA slots. Do not force the PCI card into a PC/AT slot. The PCI-8134/PCI-8134A can be used in any PCI slot.

2.3.3 Installation Procedures

Read through this manual, and setup the jumper according to your application

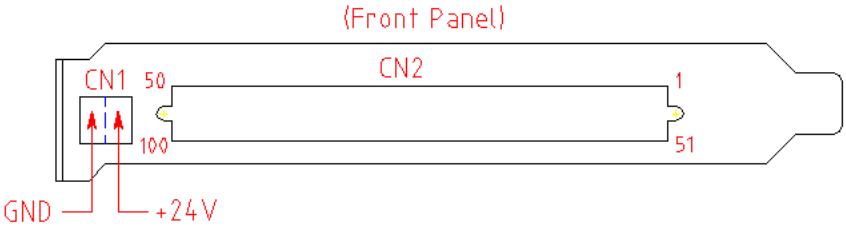
Turn off your computer, Turn off all accessories (printer, modem, monitor, etc.) connected to computer.

Remove the cover from your computer.

Select a 32-bit PCI expansion slot. PCI slots are short than ISA or EISA slots and are usually white or ivory.

Before handling the PCI-8134/PCI-8134A, discharge any static buildup on your body by touching the metal case of the computer. Hold the edge and do not touch the components.

Position the board into the PCI slot you selected.



Secure the card in place at the rear panel of the system unit using screw removed from the slot.

2.3.4 Troubleshooting:

If your system won‘t boot or if you experience erratic operation with your PCI board in place, it’s likely caused by an interrupt conflict (perhaps because you incorrectly described the ISA setup). In general, the solution, once you determine it is not a simple oversight, is to consult the BIOS documentation that comes with your system.

2.4 Software Driver Installation

Please refer to the ADLink All-in-one Compact Disc Manual to install it.

2.5 Programming Guide Installation

1) From the ADLINK All-In-One CD Choose Driver Installation>Motion Control>PCI-8134/PCI-8134A
2) Follow the procedures of the installer.
3) After installation is completed, restart Windows.

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

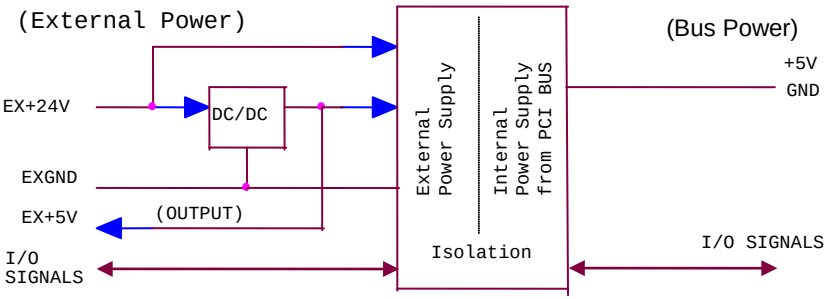
2.6 CN1 Pin Assignments: External Power Input

CN1 Pin No	Name	Description
1	EXGND	Grounds of the external power.
2	EX+24V	External power supply of +24V DC ± 5%

Note:

1. CN1 is a plug-in terminal board with no screw.
2. Be sure to use the external power supply. The +24V DC is used by external input/output signal circuit. The power circuit is configured as follows.
3. Wires for connection to CN1.
   Solid wire: φ 0.32mm to φ 0.65mm (AWG28 to AWG22)
   Twisted wire: 0.08mm2 to 0.32mm2 (AWG28 to AWG22)
   Naked wire length: 10mm standard.

The following diagram shows the external power supply system of the PCI-8134/PCI-8134A. The external +24V power must be provided, an on-board regulator generates +5V for both internal and external usage.



2.7 CN2 Pin Assignments: Main connector

The CN2 is the major connector for the motion control I/O signals.

No.	Name	I/O	Function(axis①/②)	No.	Name	I/O	Function(axis③/④)
1	EX+5V	O	+5V power supply output	51	EX+5V	O	+5V power supply output
2	EXGND		Ext. power ground	52	EXGND		Ext. power ground
3	OUT1+	O	Pulse signal (+),①	53	OUT3+	O	Pulse signal (+), ③
4	OUT1-	O	Pulse signal (-),①	54	OUT3-	O	Pulse signal (-),③
5	DIR1+	O	Dir. signal (+),①	55	DIR3+	O	Dir. signal (+), ③
6	DIR1-	O	Dir. signal (-),①	56	DIR3-	O	Dir. signal (-), ③
7	SVON1	O	Multi-purpose signal, ①	57	SVON3	O	Multi-purpose signal, ③
8	ERC1	O	Dev. ctr, clr. signal, ①	58	ERC3	O	Dev. ctr, clr. signal, ③
9	ALM1	I	Alarm signal, ①	59	ALM3	I	Alarm signal, ③
10	INP1	I	In-position signal, ①	60	INP3	I	In-position signal, ③
11	RDY1	I	Multi-purpose signal, ①	61	RDY3	I	Multi-purpose signal, ③
12	EXGND		Ext. power ground	62	EXGND		Ext. power ground
13	EA1+	I	Encoder A-phase (+), ①	63	EA3+	I	Encoder A-phase (+), ③
14	EA1-	I	Encoder A-phase (-), ①	64	EA3-	I	Encoder A-phase (-),③
15	EB1+	I	Encoder B-phase (+), ①	65	EB3+	I	Encoder B-phase (+),③
16	EB1-	I	Encoder B-phase (-), ①	66	EB3-	I	Encoder B-phase (-),③
17	EZ1+	I	Encoder Z-phase (+), ①	67	EZ3+	I	Encoder Z-phase (+),③
18	EZ1-	I	Encoder Z-phase (-), ①	68	EZ3-	I	Encoder Z-phase (-),③
19	EX+5V	O	+5V power supply output	69	EX+5V	O	+5V power supply output
20	EXGND		Ext. power ground	70	EXGND		Ext. power ground
21	OUT2+	O	Pulse signal (+), ②	71	OUT4+	O	Pulse signal (+),④
22	OUT2-	O	Pulse signal (-), ②	72	OUT4-	O	Pulse signal (-),④
23	DIR2+	O	Dir. signal (+), ②	73	DIR4+	O	Dir. signal (+),④
24	DIR2-	O	Dir. signal (-), ②	74	DIR4-	O	Dir. signal (-),④
25	SVON2	O	Multi-purpose signal, ②	75	SVON4	O	Multi-purpose signal, ④
26	ERC2	O	Dev. ctr, clr. signal, ②	76	ERC4	O	Dev. ctr, clr. signal, ④
27	ALM2	I	Alarm signal, ②	77	ALM4	I	Alarm signal, ④
28	INP2	I	In-position signal, ②	78	INP4	I	In-position signal, ④
29	RDY2	I	Multi-purpose signal, ②	79	RDY4	I	Multi-purpose signal, ④
30	EXGND		Ext. power ground	80	EXGND		Ext. power ground
31	EA2+	I	Encoder A-phase (+), ②	81	EA4+	I	Encoder A-phase (+), ④
32	EA2-	I	Encoder A-phase (-), ②	82	EA4-	I	Encoder A-phase (-), ④
33	EB2+	I	Encoder B-phase (+), ②	83	EB4+	I	Encoder B-phase (+), ④
34	EB2-	I	Encoder B-phase (-), ②	84	EB4-	I	Encoder B-phase (-), ④

35	EZ2+	I	Encoder Z-phase (+), ②	85	EZ4+	I	Encoder Z-phase (+), ④
36	EZ2-	I	Encoder Z-phase (-), ②	86	EZ4-	I	Encoder Z-phase (-), ④
37	PEL1	I	End limit signal (+), ①	87	PEL3	I	End limit signal (+), ③
38	MEL1	I	End limit signal (-), ①	88	MEL3	I	End limit signal (-), ③
39	PSD1	I	Ramp-down signal (+), ①	89	PSD3	I	Ramp-down signal (+), ③
40	MSD1	I	Ramp-down signal (-), ①	90	MSD3	I	Ramp-down signal (-), ③
41	ORG1	I	Origin signal, ①	91	ORG3	I	Origin signal, ③
42	EXGND		Ext. power ground	92	EXGND		Ext. power ground
43	PEL2	I	End limit signal (+), ②	93	PEL4	I	End limit signal (+), ④
44	MEL2	I	End limit signal (-), ②	94	MEL4	I	End limit signal (-), ④
45	PSD2	I	Ramp-down signal (+), ②	95	PSD4	I	Ramp-down signal (+), ④
46	MSD2	I	Ramp-down signal (-), ②	96	MSD4	I	Ramp-down signal (-), ④
47	ORG2	I	Origin signal, ②	97	ORG4	I	Origin signal, ④
48	EXGND		Ext. power ground	98	EXGND		Ext. power ground
49	EXGND		Ext. power ground	99	EX+24V	I	Ext. power supply, +24V
50	EXGND		Ext. power ground	100	EX+24V	I	Ext. power supply, +24V

2.8 CN3 Pin Assignments: Manual Pulser Input

The signals on CN3 is for manual pulser input.

No.	Name	Function(Axis)
1	GND	Bus power ground
2	PB4	Pulser B-phase signal input, ④
3	PA4	Pulser A-phase signal input, ④
4	PB3	Pulser B-phase signal input, ③
5	PA3	Pulser A-phase signal input, ③
6	+5V	Bus power, +5V
7	GND	Bus power ground
8	PB2	Pulser B-phase signal input, ②
9	PA2	Pulser A-phase signal input, ②
10	PB1	Pulser B-phase signal input, ①
11	PA1	Pulser A-phase signal input, ①
12	+5V	Bus power, +5V

Note: +5V and GND pins are directly given by the PCI-Bus power. Therefore, these signals are not isolated.

2.9 CN4 Pin Assignments: Simultaneous Start/Stop

The signals on CN3 is for simultaneously start/stop signals for multiple axes and multiple cards.

No.	Name	Function(Axis)
1	GND	Bus power ground
2	STP	Simultaneous stop signal input/output
3	STA	Simultaneous start signal input/output
4	STP	Simultaneous stop signal input/output
5	STA	Simultaneous start signal input/output
6	+5V	Bus power, +5V

Note: +5V and GND pins are directly given by the PCI Bus power.

2.10 Jumper Setting

The J1~J8 is used to set the signal type of the pulse output signals (DIR and OUT). The output signal type could be differential line driver output or open collector output. Please refer to section 3.1 for details of the jumper setting. The default setting is the differential line driver mode.

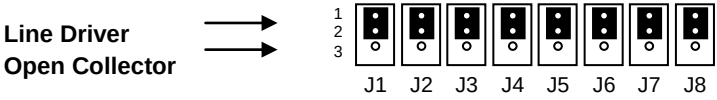


Figure 2.3 Illustration of PCI-8134 jumpers

Figure 2.4 Illustration of PCI-8134A jumpers

2.11 Switch Setting

The switch S1 is used to set the EL limit switch's type. The default setting of EL switch type is "normal open" type limit switch (or "A" contact type). The switch on is to use the "normal closed" type limit switch (or "B" contact type). The default setting is set as normal open type.

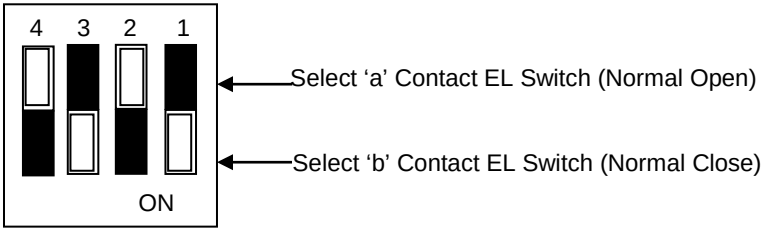


Figure 2.5 Placement of S1 Switch on Board of PCI-8134

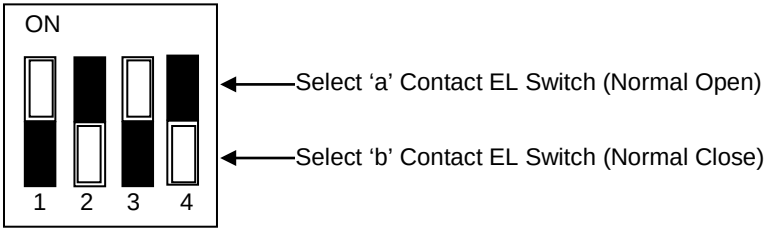


Figure 2.6 Placement of S1 Switch on Board of PCI-8134A

3

Signal Connections

The signal connections of all the I/O signals are described in this chapter. Please refer the contents of this chapter before wiring the cable between the PCI-8134/PCI-8134A and the motor drivers.

This chapter contains the following sections:

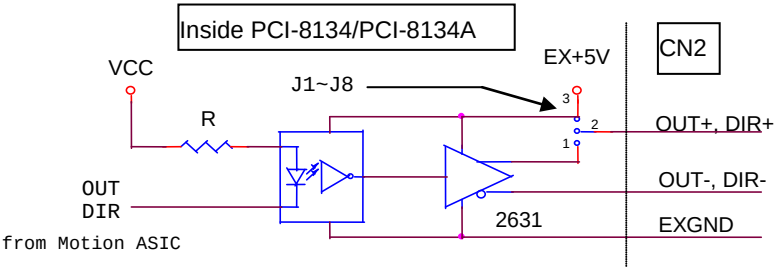
3.1 Pulse Output Signals OUT and DIR

There are 4-axis pulse output signals on PCI-8134/PCI-8134A. For every axis, two pairs of OUT and DIR signals are used to send the pulse train and to indicate the direction. The OUT and DIR signals can also be programmed as CW and CCW signals pair, refer to section 4.1.1 for details of the logical characteristics of the OUT and DIR signals. In this section, the electronic characteristics of the OUT and DIR signals are shown. Each signal consists of a pair of differential signals. For example, the OUT2 is consisted of OUT2+ and OUT2- signals. The following table shows all the pulse output signals on CN2.

CN2 Pin No.	Signal Name	Description	Axis #
3	**OUT1+**	Pulse signals (+)	①
4	**OUT1-**	Pulse signals (-)	①
5	**DIR1+**	Direction signal(+)	①
6	**DIR1-**	Direction signal(-)	①
21	**OUT2+**	Pulse signals (+)	②
22	**OUT2-**	Pulse signals (-)	②
23	**DIR2+**	Direction signal(+)	②
24	**DIR2-**	Direction signal(-)	②
53	**OUT3+**	Pulse signals (+)	③
54	**OUT3-**	Pulse signals (-)	③
55	**DIR3+**	Direction signal(+)	③
56	**DIR3-**	Direction signal(-)	③
71	**OUT4+**	Pulse signals (+)	④
72	**OUT4-**	Pulse signals (-)	④
73	**DIR4+**	Direction signal(+)	④
74	**DIR4-**	Direction signal(-)	④

The output of the OUT or DIR signals can be configured by jumpers as either the differential line driver or open collector output. You can select the output mode either by closing breaks between 1 and 2 or 2 and 3 of jumpers J1~J8 as follows.

Output Signal	For differential line driver output, close a break between 1 and 2 of	For open collector output, close a break between 2 and 3 of:
OUT1-	J1	J1
DIR1-	J2	J2
OUT2-	J3	J3
DIR2-	J4	J4
OUT3-	J5	J5
DIR3-	J6	J6
OUT4-	J7	J7
DIR4-	J8	J8



The **default** setting of OUT and DIR signals are the as differential line driver mode.

The following wiring diagram is for the OUT and DIR signals of the 4 axes.

NOTE: If the pulse output is set to the open collector output mode, the OUT- and DIR- are used to send out signals. Please take care that the current sink to OUT- and DIR- pins must not exceed 20mA. The current may provide by the EX+5V power source, however, please note that the maximum capacity of EX+5V power is 500mA.

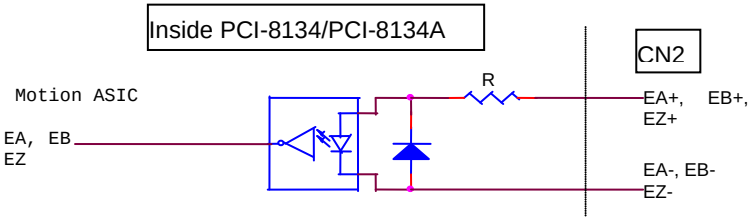
3.2 Encoder Feedback Signals EA, EB and EZ

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

CN2 Pin No	Signal Name	Axis #	CN2 Pin No	Signal Name	Axis #
13	EA1+	①	63	EA3+	③
14	EA1-	①	64	EA3-	③
15	EB1+	①	65	EB3+	③
16	EB1-	①	66	EB3-	③
31	EA2+	②	81	EA4+	④
32	EA2-	②	82	EA4-	④
33	EB2+	②	83	EB4+	④
34	EB2-	②	84	EB4-	④

CN2 Pin No	Signal Name	Axis #	CN2 Pin No	Signal Name	Axis #
17	EZ1+	①	67	EZ3+	③
18	EZ1-	①	68	EZ3-	③
35	EZ2+	②	85	EZ4+	④
36	EZ2-	②	86	EZ4-	④

The input circuits of the EA, EB, EZ signals are shown as follows.



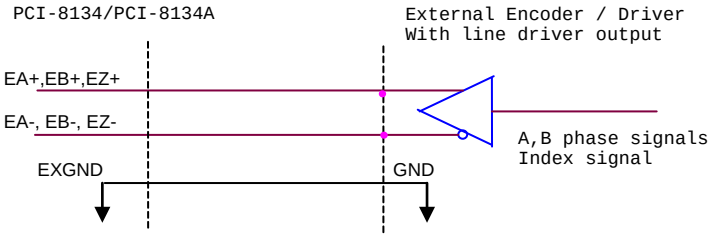
Please note that the voltage across every differential pair of encoder input signals (EA+, EA-), (EB+, EB-) and (EZ+, EZ-) should be at least 3.5V or higher. Therefore, you have to take care of the driving capability when connecting with the encoder feedback or motor driver feedback. The

differential signal pairs will be converted to digital signal EA, EB and EZ to connect to Motion ASIC.

Here are two examples of connecting the input signals with the external circuits. The input circuits can connect to the encoder or motor driver, which are equipped with: (1) differential line driver or (2) open collector output.

◆ **Connection to Line Driver Output**

To drive the PCI-8134/PCI-8134A encoder input, the driver output must provide at least 3.5V across the differential pairs with at least 6 mA driving capability. The ground level of the two sides must be tight together too.



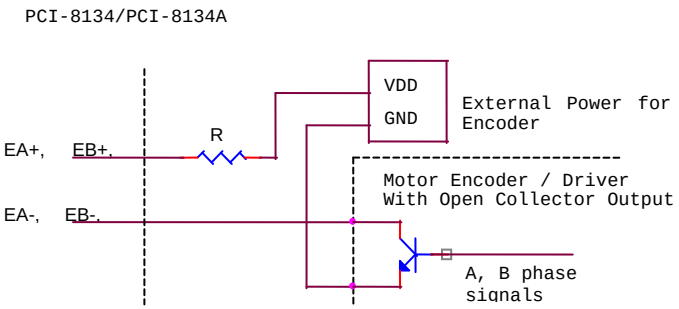
◆ **Connection to Open Collector Output**

To connect with open collector output, an external power supply is necessary. Some motor drivers also provide the power source. The connection between PCI-8134/PCI-8134A, encoder, and the power supply is shown in the following diagram. Please note that the external current limit resistor R is necessary to protect the PCI-8134/PCI-8134A input circuit. The following table lists the suggested resistor value according to the encoder power supply.

Encoder Power(VDD)	External Resistor R
+5V	0Ω (None)
+12V	1.8kΩ
+24V	4.3kΩ

If=6mA max.

For more detail operation of the encoder feedback signals, please refer to section 4.4.

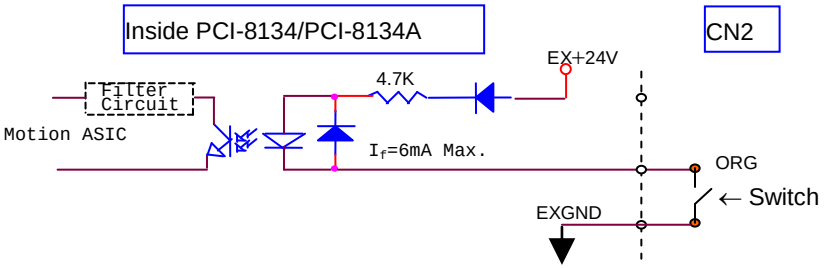


3.3 Origin Signal ORG

The origin signals (ORG1~ORG4) are used as input signals for origin of the mechanism. The following table lists the relative signal name, pin number, and the axis number.

CN2 Pin No	Signal Name	Axis #
41	ORG1	①
47	ORG2	②
91	ORG3	③
97	ORG4	④

The input circuits of the ORG signals are shown as following. Usually, a limit switch is used to indicate the origin of one axis. The specifications of the limit switches should with contact capacity of +24V, 6mA minimum. An internal filter circuit is used to filter out the high frequency spike, which may cause wrong operation.



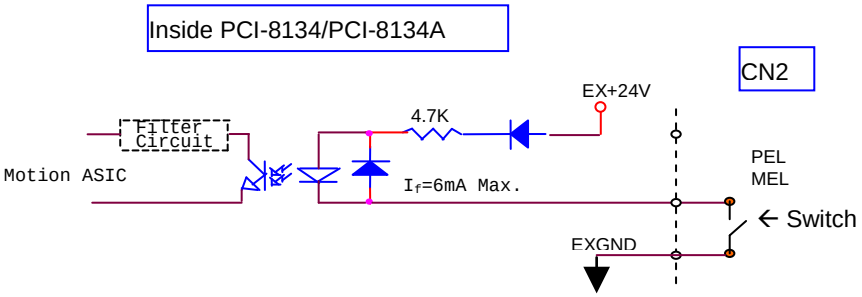
When the motion controller is operated at the home return mode, the ORG signal is used to stop the control output signals (OUT and DIR). For the detail operation of the ORG, please refer to section 4.3.3

3.4 End-Limit Signals PEL and MEL

There are two end-limit signals PEL and MEL for one axis. PEL indicates end limit signal in plus direction and MEL indicates end limit signal in minus direction. The relative signal name, pin number and axis number are shown in the following table.

CN2 Pin No	Signal Name	Axis #	CN2 Pin No	Signal Name	Axis #
37	PEL1	①	87	PEL3	③
38	MEL1	①	88	MEL3	③
43	PEL2	②	93	PEL4	④
44	MEL2	②	94	MEL4	④

The signals connection and relative circuit diagram is shown in the following diagram. The external limit switches featuring a contact capacity of +24V, 6mA minimum. You can use either 'A-type' (normal open) contact switch or 'B-type' (normal closed) contact switch by setting the DIP switch S1. The PCI-8134/PCI-8134A is delivered with all bits of S1 set to OFF, refer to section 2.10. For the details of the EL operation, please refer to section 4.3.2.

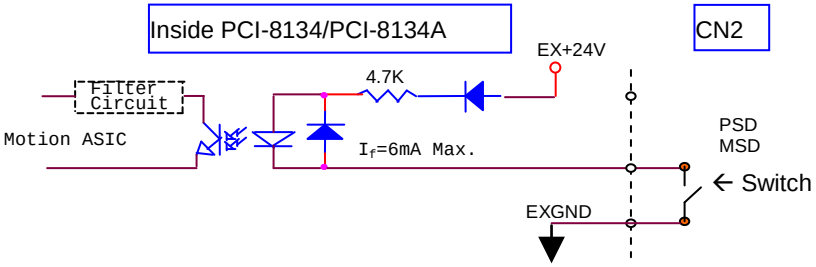


3.5 Ramping-down Signals PSD and MSD

There are two ramping-down (Slow–Down) signals PSD and MSD for one axis. The relative signal name, pin number and axis number are shown in the following table.

CN2 Pin No	Signal Name	Axis #
39	PSD1	①
40	MSD1	①
45	PSD2	②
46	MSD2	②
89	PSD3	③
90	MSD3	③
95	PSD4	④
96	MSD4	④

The signals connection and relative circuit diagram is shown in the following diagram. Usually, limit switches are used to generate the slow–down signals to make motor operating in a slower speed. For more details of the SD operation, please refer to section 4.3.1.

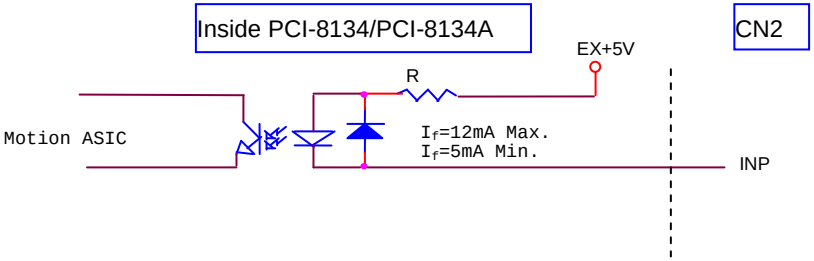


3.6 In-position Signal INP

The in-position signals INP from the servo motor driver indicate the deviation error is zero, that is the servo position error is zero. The relative signal name, pin number and axis number are shown in the following table.

CN2 Pin No	Signal Name	Axis #
10	INP1	①
28	INP2	②
60	INP3	③
78	INP4	④

The input circuit of the INP signals are shown in the following diagram.



The in-position signals are usually from servomotor drivers, which usually provide open collector output signals. The external circuit must provide at least 5 mA current sink capability to drive the INP signal active. For more details of the INP signal operating, please refer to section 4.2.1.

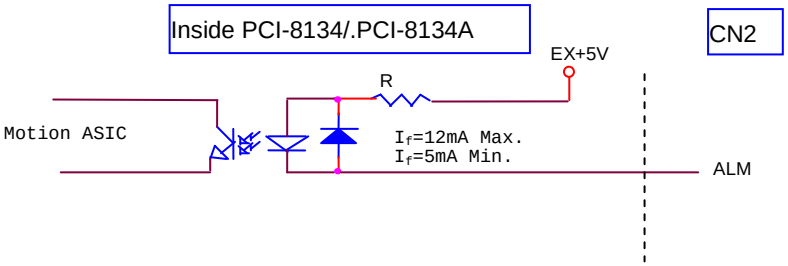
3.7 Alarm Signal ALM

The alarm signal ALM is used to indicate the alarm status from the servo driver. The relative signal name, pin number and axis number are shown in the following table.

CN2 Pin No	Signal Name	Axis #
9	ALM1	①
27	ALM2	②
59	ALM3	③
77	ALM4	④

The input circuit of alarm circuit is shown in the following diagram. The ALM signals are usually from servomotor drivers, which usually provide open

collector output signals. The external circuit must provide at least 5 mA current sink capability to drive the ALM signal active. For more details of the ALM operation, please refer to section 4.2.2.



3.8 Deviation Counter Clear Signal ERC

The deviation counter clear signal (ERC) is active in the following 4 situations:

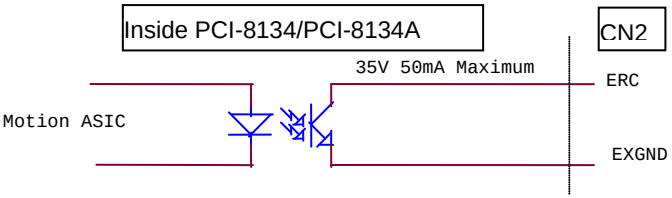
1. home return is complete;
2. the end-limit switch is active;
3. an alarm signal stops OUT and DIR signals;
4. an emergency stop command is issued by software (operator).

The relative signal name, pin number and axis number are shown in the following table.

CN2 Pin No	Signal Name	Axis #
8	ERC1	①
26	ERC2	②
58	ERC3	③
76	ERC4	④

The ERC signal is used to clear the deviation counter of servomotor driver. The ERC output circuit is in the open collector with maximum 35 V external

power at 50mA driving capability. For more details of the ERC operation, please refer to section 4.2.3.

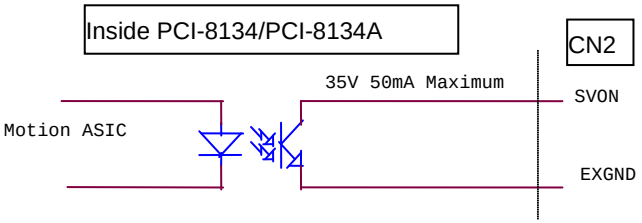


3.9 General-purpose Signal SVON

The SVON signals can be used as servomotor-on control or general-purpose output signals. The relative signal name, pin number and axis number are shown in the following table.

CN2 Pin No	Signal Name	Axis #
7	SVON1	①
25	SVON2	②
57	SVON3	③
75	SVON4	④

The output circuit of SVON signal is shown in the following diagram.

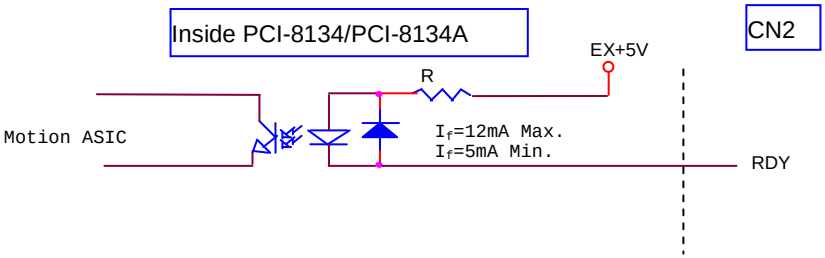


3.10 General-purpose Signal RDY

The RDY signals can be used as motor driver ready input or general–purpose input signals. The relative signal name, pin number and axis number are shown in the following table.

CN2 Pin No	Signal Name	Axis #
11	RDY1	①
29	RDY2	②
61	RDY3	③
71	RDY4	④

The input circuit of RDY signal is shown in the following diagram

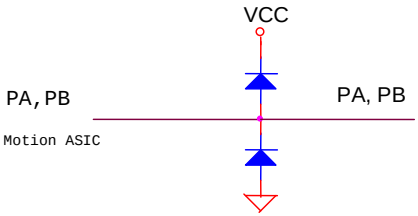


3.11 Pulser Input Signals PA and PB

The PCI-8134/PCI-8134A can accept the input signals from pulser signals through the following pins of connector CN3. The pulser's behavior is as an encoder. The signals are usually used as generate the position information which guide the motor to follow.

CN3 Pin No	Signal Name	Axis #	CN3 Pin No	Signal Name	Axis #
2	PA1	①	8	PA3	③
3	PB1	①	9	PB3	③
4	PA2	②	10	PA4	④
5	PB2	②	11	PB4	④

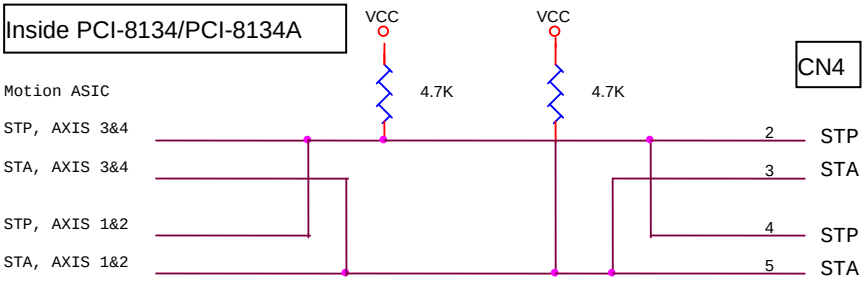
PA and PB pins of connector CN3 are directly connected to PA and PB pins of PCL5023. The interfac circuits are shown as follows.



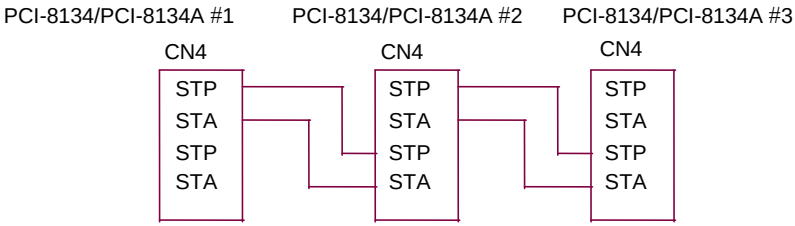
If the signal voltage of pulser is not +5V or if the pulser is distantly placed, it is recommended to put a photo coupler or line driver in between. Also, +5V and GND power lines of CN3 are direct from the PCI bus. Please carefully use these signals because they are not isolated.

3.12 Simultaneously Start/Stop Signals STA and STP

The PCI-8134/PCI-8134A provides the STA and STP signals, which enable simultaneous start/stop of motions on multiple axes. The STA and STP signals are on the CN4.



In order to implement axes synchronous control between different cards, both PCI-8134 and PCI-8134A are able to synchronize the axes control through simultaneous control signals, STA and STP. User is able to connect each STA and STP signal via CN4 connector as the following illustration. Also user would use external signals to trigger the simultaneous axes control.


PCI-8134/PCI-8134A #1
CN4
STP
STA
STP
STA
PCI-8134/PCI-8134A #2
CN4
STP
STA
STP
STA
PCI-8134/PCI-8134A #3
CN4
STP
STA
STP
STA

4

Operations

This chapter describes detailed operation of the PCI-8134/PCI-8134A card. Contents of the following sections are as following.

Section 4.1: The motion control modes
Section 4.2: The motor driver interface (INP, ERC, ALM, SVON, RDY)
Section 4.3: The limit switch interface and I/O status (SD, EL, ORG)
Section 4.4: The encoder feedback signals (EA, EB, EZ)
Section 4.5: Multiple PCI-8134/PCI-8134A cards operation.
Section 4.6: Change Speed on the Fly
Section 4.7: Interrupt Control

4.1 Motion Control Modes

In this section, the pulse output signals' configurations, and the following motion control modes are described.

- Constant velocity motion for one axis
- Trapezoidal motion for one axis
- S-Curve profile motion for one axis
- Linear interpolation for two axes
- Home return mode for one axis
- Manual pulser mode for one axis

4.1.1 Pulse Command Output

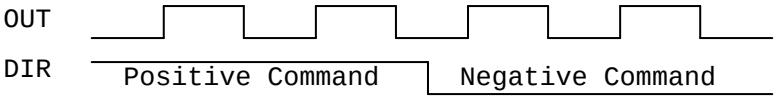
The PCI-8134/PCI-8134A uses pulse command to control the servo / stepper motors via the drivers. The pulse command consists of two signals: OUT and DIR. There are two command types: (1) single pulse output mode (OUT/DIR); and (2) dual pulse output mode (CW/CCW type pulse output). The software function: set_pls_outmode() is used to program the pulse command type. The modes vs. signal type of OUT and DIR pins are as following table:

Mode	Output of OUT pin	Output of DIR pin
Dual pulse output	Pulse signal in plus (or CW) direction	Pulse signal in minus (or CCW) direction
Single pulse output	Pulse signal	Direction signal (level)

The interface characteristics of these signals could be differential line driver or open collector output. Please refer to section 3.1 for the jumper setting of signal types.

Single Pulse Output Mode (OUT/DIR Mode)

In this mode, the OUT signal is represent the pulse (position or velocity) command. The numbers of OUT pulse represent the motion command for relative “distance” or “position”, the frequency of the OUT pulse represents the command for “speed” or “velocity”. The DIR signal represents direction command of the positive (+) or negative (-). This mode is the most common

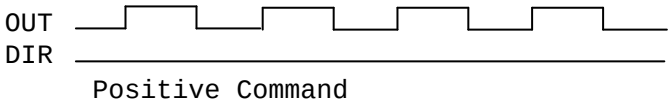


used mode. The following diagram shows the output waveform.

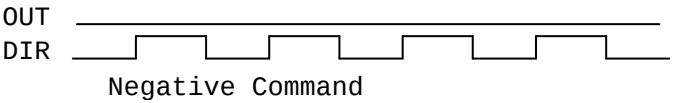
Dual Pulse Output Mode (CW/CCW Mode)

In this mode, the waveform of the OUT and DIR pins represents CW (clockwise) and CCW (counter clockwise) pulse output respectively. Pulses output from CW pin makes motor move in positive direction, whereas pulse output from CCW pin makes motor move in negative direction. The

following diagram shows the output waveform of positive (plus,+) command



and negative (minus,-) command.



Relative Function:

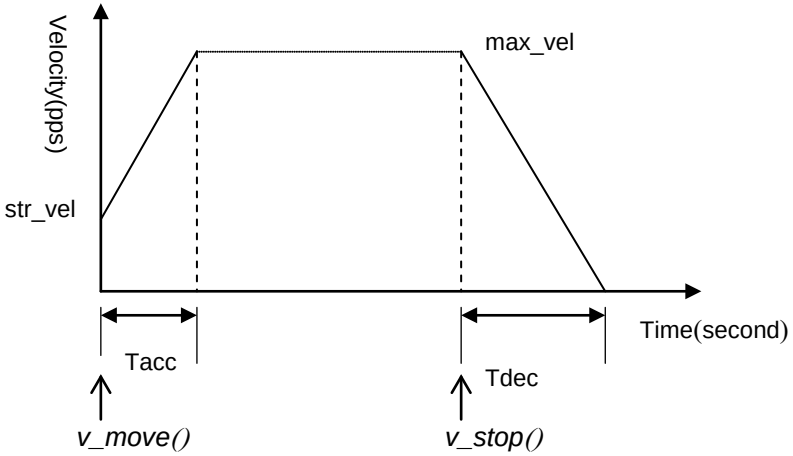
set_pls_optmode(): Refer to section 6.4

4.1.2 Constant Velocity Motion

This mode is used to operate one axis motor at constant velocity motion. The output pulse accelerates from a starting velocity (str_vel) to the specified constant velocity (max_vel). The **v_move()** function is used to accelerate constantly while the **sv_move()** function is to accelerate according to S-curve (constant jerk). The pulse output rate will keep at maximum velocity until another velocity command is set or stop command is issued. The **v_change()** is used to change speed during moving. The ***v_stop()*** function is used to decelerate the motion to zero velocity (stop). The velocity profile is shown as following. **Note** that v_stop() function can be also be applied to stop outputting command pulses during **Preset Mode** (both trapezoidal and S-curve Motion) , **Home Mode** or **Manual Pulser Mode** operations**.**

Relative Functions:

v_move( ), v_stop( ), sv_move(): Refer to section 6.5



4.1.3 Trapezoidal Motion

This mode is used to move one axis motor to a specified position (or distance) with a trapezoidal velocity profile. Single axis is controlled from point to point. An absolute or relative motion can be performed. In absolute mode, the target position is assigned. In relative mode, the target displacement is assigned. In both absolute and relative mode, the acceleration and the deceleration can be different. The **motion_done()** function is used to check whether the movement is complete.

The following diagram shows the trapezoidal profile. There are 9 relative functions. In the **a_move()**, **ta_move()** and **start_a_move()**, **start_ta_move()** functions, the absolute target position must be given in the unit of pulse. The physical length or angle of one movement is dependent on the motor driver and the mechanism (includes the motor). Since absolute move mode needs the information of current actual position, so "External encoder feedback (EA, EB pins)" must be enabled in **set_cnt_src()** function. And the ratio between command pulses and external feedback pulse input must be appropriately set by **set_move_ratio()** function.

In the **r_move()**, **t_move()** and **start_r_move()**, **start_t_move()** functions, the relative displacement must be given in the unit of pulse. Unsymmetrical trapezoidal velocity profile (Tacc is not equal Tdec) can be specified in **ta_move()** and **t_move()** functions; where symmetrical profile (Tacc = Tdec) can be specified in **a_move()** and **r_move()** functions

The str_vel and max_vel parameters are given in the unit of pulse per second (pps). The Tacc and Tdec parameters are given in the unit of second represent accel./decel. time respectively. You have to know the physical meaning of "one movement" to calculate the physical value of the relative velocity or acceleration parameters. The following formula gives the basic relationship between these parameters.

max_vel = str_vel + accel*Tacc;

str_vel = max_vel + decel *Tdec;

where accel/decel represents the acceleration/deceleration rate in unit of pps/sec. The area inside the trapezoidal profile represents the moving distance.

The unit of velocity setting is pulses per second (pps). Usually, the unit of velocity in the manual of motor or driver is in rounds per minute (rpm). A simple conversion is necessary to match between these two units. Here we use a example to illustrate the conversion.

For example:

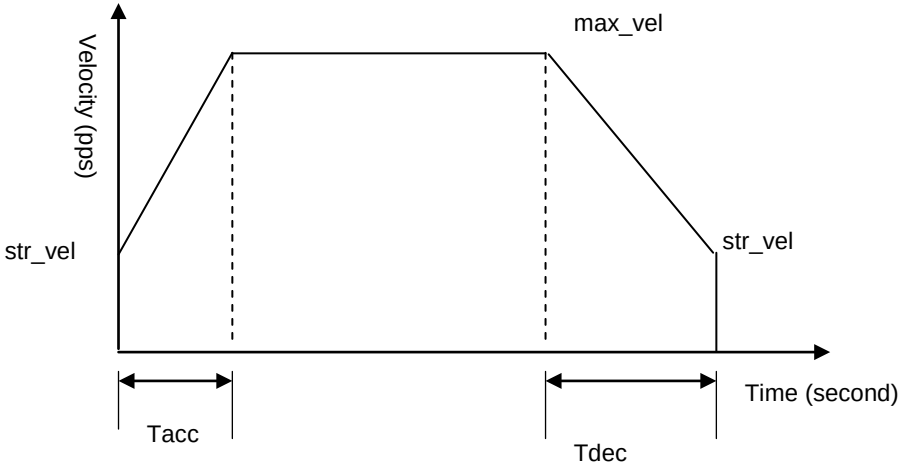
A servo motor with a AB phase encoder is used for a X-Y table. The resolution of encoder is 2000 counts per phase. The maximum rotating speed of motor is designed to be 3600 rpm. What is the maximum pulse command output frequency that you have to set on PCI-8134/PCI-8134A?

Answer:

max_vel = 3600/60*2000*4

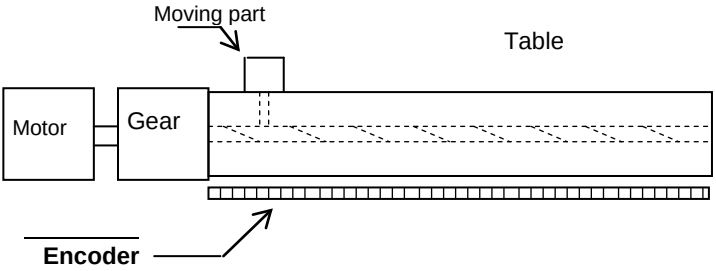
= 48000pps

The reason why *4 is because there are four states per AB phase (See Figures in Section 4.4).



Usually, the axes need to set the move ratio if their mechanical resolution is different from the resolution of command pulse. For example, if an incremental type encoder is mounted on the working table to measure the actual position of moving part. A servomotor is used to drive the moving part through a gear mechanism. The gear mechanism is used to convert the rotating motion of motor into linear motion.(see the following diagram). If the resolution of motor is 8000 pulses/round. The resolution of gear mechanism is 100 mm/round.(i.e., part moves 100 mm if motor turns one round). Then the resolution of command pulse will be 80 pulses/mm. The resolution of encoder mounting on the table is 200 pulses/mm. Then users have to set the move ratio as 200/80=2.5 by the function:

set_move_ratio(axis, 2.5);



If this ratio is not set before issuing the start moving command, it will cause problems when running in “Absolute Mode”. Because the PCI-8134/PCI-8134A can’t recognize the actual absolute position during motion.

Relative Functions:

a_move(), r_move(), t_move(), ta_move(), start_a_move(), start_r_move(), start_t_move(), start_ta_move() Refer to section 6.6.
motion_done(): Refer to section 6.13.
set_cnt_src(): Refer to section 6.4.
set_move_ratio(): Refer to section 6.10.

4.1.4 S-curve Profile Motion

This mode is used to move one axis motor to a specified position (or distance) with a S-curve velocity profile. S-curve acceleration profiles are useful for both stepper and servo motors. The smooth transitions between the start of the acceleration ramp and the transition to the constant velocity produce less wear and tear than a trapezoidal profile motion. The smoother performance increases the life of the motors and mechanics of a system.

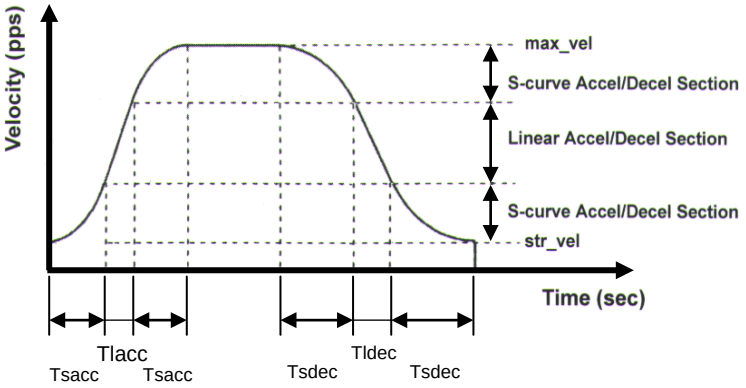
There are several parameters needed to be set in order to make a S-curve move. They are:

pos: target position in absolute mode;
dist: moving distance in relative mode;
str_vel : specify the start velocity;
max_vel : specify the maximum velocity;
Tlacc: specify the time for linear acceleration section
(constant acceleration).
Tsacc: specify the time for S-curve acceleration section

(constant jerk).

Tldec: specify the time for linear deceleration section
(constant deceleration).

Tsdec: specify the time for S-curve deceleration section



(constant jerk).

Total time of acceleration is: Tlacc+2Tsacc. The following formula gives the basic relationship between these parameters.

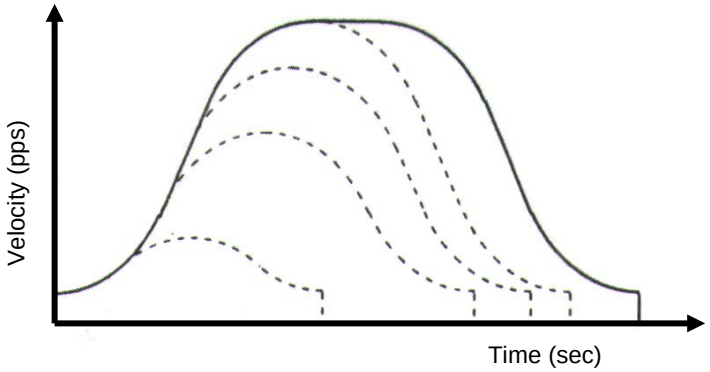
```
max_vel = str_vel + accel*(Tlacc+Tsacc);
str_vel  = max_vel + decel *(Tldec+Tsdec);
accel = Tsacc * jerk1;
decel = Tsdec * jerk2;
```

where accel/decel represents the acceleration/deceleration rate at linear accel./decel. section and are in unit of pps/sec. jerk1, jerk2 are in unit of pps/sec^2. The minimum value for setting time of accel./decel. should be 0.

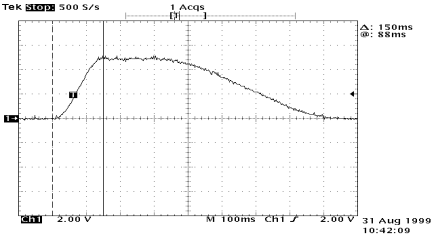
The S-curve profile motion functions are designed to always produce smooth motion. If the time for linear/S-Curve acceleration parameters combined with the final position don't allow an axis to reach the maximum

velocity(i.e.: the moving distance is too small to reach max_vel), the maximum velocity is automatically lowered and

smooth accel./decel. is made (see the following Figure). This means that with moves that don't reach maximum velocity may cause longer than expected move times. In such a case, the smaller the moving distance, the shorter the linear accel./decel. section becomes and the S-curve section is not reduced unless the linear section is decreased to 0.



The following two graphs show the results of experiments after executing the unsymmetrical absolute S-curve motion command. Graph1 is the typical result. of S-curve velocity profile. Graph2 is obtained when the amount of command pulses is failed to let the velocity reach the designated maximum velocity. The PCI-8134/PCI-8134A automatically lower the maximum velocity thus provide a smooth velocity profile.



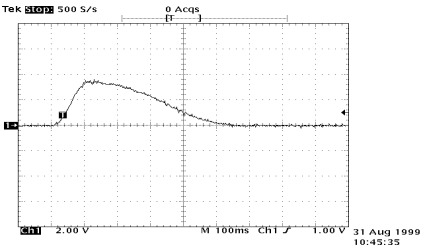
Command of Graph1:

start_tas_move(axis, 500000, 100, 1000000, 0.05, 0.05, 0.2, 0.2);

The total accelerating time = 0.05+2*0.05 = 0.15 (second).
Total decelerating time = 0.2+2*0.2 = 0.6 (second).

Command of Graph2:

start_tas_move(axis, 200000, 100, 1000000, 0.05, 0.05, 0.2, 0.2);

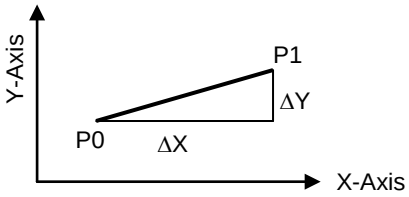


Relative Functions:

s_move(), rs_move(), tas_move(), start_s_move(), start_rs_move(), start_tas_move() Refer to section 6.7
motion_done(): Refer to section 6.13

4.1.5 Linear Interpolated Motion

In this mode, two axes (″X and Y″ or ″Z and U″ axes) is controlled by linear interpolation or circular interpolation by designating the number of pulses respectively. ″Interpolation between two axes″ means the two axes start simultaneously, and reach their ending points at the same time. For example, in the Figure below, we want to move the axes from P0 to P1, and hope the two axes start and stop simultaneously at a period of time Δt. Then the moving speed along X-axis and Y-axis will beΔX/Δt., ΔY/Δt. respectively.



The axis with larger numbers of moving pulses is the main axis, and the other axis is the secondary axis. When both axes are set at the same amount of pulses, the 'X' or 'Z' is the main axis. The speed relation between main and secondary axes is as follows:

$$\frac{\sqrt{(\text{Set number of pulses for main axis})^2 + (\text{Set number of pulses for slave axis})^2}}{\text{Set number of pulses for main axis}}$$

Composite Speed = Speed of main axis x

Relative Functions:

move_xy(), move_zu(),Refer to section 6.9

set_move_speed(), set_move_accel(),set_move_ratio(): Refer to section 6.10

$$\text{or} = \text{Speed of main axis} \times \sqrt{1+\left(\frac{\text{Set number of pulses for slave axis}}{\text{Set number of pulses for main axis}}\right)^2}$$

4.1.6 Home Return Mode

In this mode, you can let the PCI-8134/PCI-8134A output pulses until the conditions to complete the home return is satisfied after writing the **home_move()** command. Finish of home return can be checked by **motion_done()** function. Or you can check finish of home return accompanied with the interrupt function by setting bit 5 of ***int_factor*** to 1 in ***set_int_factor()*** function.

Moving direction of motors in this mode is determined by the sign of velocity parameter in **home_move()** function. A **v_stop()** command during returning home can stop OUT and DIR from outputting pulses.

Before writing **home_move()** command, configuration must be set by **set_home_config()** function. . See also Section 4.3.3 for further description. There are total three home return modes can be selected by

setting home_mode parameter in **set_home_config()** function. The meaning of Home_mode will be described as the following:

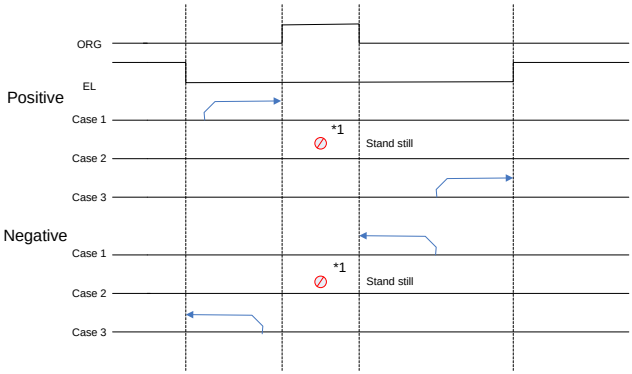
CAUTION

Due to differences between the motion chipsets of PCI-8134 and PCI-8134A, behaviour of home mode 0 and 1 will be inconsistent as performed previously. Please note differences in timing charts of each home mode for both PCI-8134 and PCI-8134A when user wants to use PCI-8134A instead of PCI-8134 with same home function. To ensure the accuracy of home move process, the motion chipset on PCI-8134A commands backward motion and stops at the edge of ORG or EZ precisely.

PCI-8134 Home Mode 0 & Home Mode 1

The timing charts of Home Mode 0 and 1 of PCI-8134/PCI-8134A follow.

PCI-8134 Home Mode 0 + ORG DO not latch



*1 In the case 2 of PCI-8134 Home Mode 0, The axis will stand still and reset counter to 0 and issue home interrupt after user commanded a home move operation.

PCI-8134 Home Mode 0 + ORG do not latch @ 8134.DLL /8134A.DLL

While the motion hits the edge of ORG or EL				
	set_cnt_src()=0 (internal)		set_cnt_src ()=1 (external)	
	get_command()	get_position()	get_command()	get_position()
Case 1	Doesn't change	Reset to 0	Doesn't change	Reset to 0
Case 2	Doesn't change	Reset to 0 (stand still)	Doesn't change	Reset to 0 (stand still)
Case 3	Doesn't change	Stop at a EL position	Doesn't change	Stop at a EL position

Counter status after Home Move Completed (Motion Done)					
	set_cnt_src()=0 (internal)		set_cnt_src ()=1 (external)		Interrupt?
	get_command()	get_position()	get_command()	get_position()	
Case 1	**Doesn't change**	**Remain 0**	**Doesn't change**	**Stop at a deceleration position**	Home Int
Case 2	**Doesn't change**	**Reset to 0**	**Doesn't change**	**Reset to 0**	Home Int
Case 3	**Doesn't change**	**Stop at a EL position**	**Doesn't change**	**Stop at a EL position**	EL Int

- Home point is at the first edge of ORG signal when home move executing. At left or right side of edge depends on home move direction.

- If axis is not at ORG, the axis will search the edge of ORG as home point.

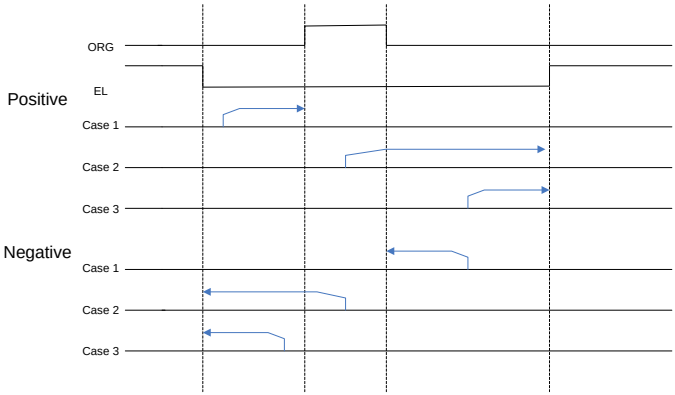
- In Case 1, the axis is stopped immediately when motion detected the edge of ORG signal but it might stop at anywhere within the range of ORG signal that means the home position is inaccurate after home move function was executed many times.

- The feedback counter of PCI-8134 will be reset to zero while the motion is hitting the edge of ORG signal.

- In Case 2, the axis will stand still and reset counter to 0 and issue home interrupt.

- After normal home finished like case 1, users have to copy to position value to command counter and target position counter at the same time.

PCI-8134 Home Mode 0 + ORG latch



PCI-8134 Home Mode 0 + ORG Latch @ 8134.DLL/8134A.DLL

While the motion hits the edge of ORG or EL				
	set_cnt_src()=0 (internal)		set_cnt_src ()=1 (external)	
	get_command()	get_position()	get_command()	get_position()
Case 1	Doesn't change	Reset to 0	Doesn't change	Reset to 0
Case 2	Doesn't change	Stop at a EL position	Doesn't change	Stop at a EL position
Case 3	Doesn't change	Stop at a EL position	Doesn't change	Stop at a EL position

Counter status after Home Move Completed (Motion Done)					
	set_cnt_src()=0 (internal)		set_cnt_src ()=1 (external)		Interrupt?
	get_command()	get_position()	get_command()	get_position()	
Case 1	Doesn't change	Remain 0	Doesn't change	Stop at a deceleration position	Home Int
Case 2	Doesn't change	Stop at a EL position	Doesn't change	Stop at a EL position	EL Int
Case 3	Doesn't change	Stop at a EL position	Doesn't change	Stop at a EL position	EL Int

- Home point is at the first edge of ORG signal when home move executing. At left or right side of edge depends on home move direction.

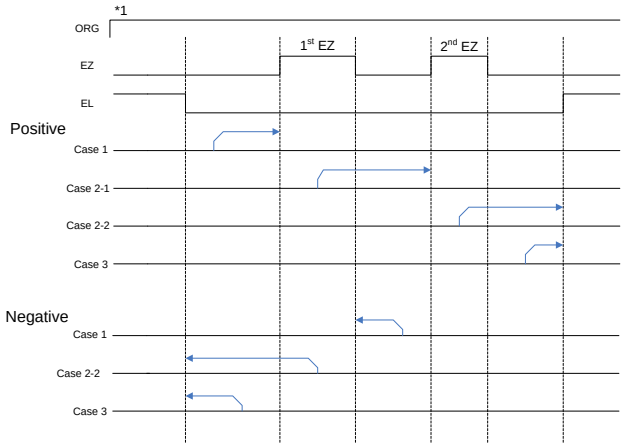
- If axis is not on ORG, the axis will search the edge of ORG as home point.

- In Case 1, the axis will be stopped immediately when axis detected the edge of ORG signal but it might stop at anywhere within the range of ORG signal that means the home position is inaccurate after home move was executed many times.

- The feedback counter of PCI-8134 will be reset to zero while the motion is hitting the edge of ORG signal.

- In Case 2 & 3, the axis will hit and then stop at the edge of EL signal anyway because the first edge of ORG signal locates behind the start point of axis that means the axis won't detect the edge of ORG signal anymore.

PCI-8134 Home Mode 1 + ORG Do not latch



*1 Once user selected "ORG DO NOT LATCH" Mode and pull-up ORG signal all the time that the PCI-8134 will search first EZ signal edge then stop immediately once user issued home move command.

PCI-8134 Home Mode 1 + ORG do not latch @ 8134A.DLL /8134.DLL

While the motion hits the edge of EZ or EL				
	set_cnt_src()=0 (internal)		set_cnt_src ()=1 (external)	
	get_command()	get_position()	get_command()	get_position()
Case 1	Doesn't change	Reset to 0	Doesn't change	Reset to 0
Case 2-1	Doesn't change	Reset to 0	Doesn't change	Reset to 0
Case 2-2	Doesn't change	Stop at a EL position	Doesn't change	Stop at a EL position
Case 3	Doesn't change	Stop at a EL position	Doesn't change	Stop at a EL position

Counter status after Home Move Completed (Motion Done)					
	Set_cnt_src()=0 (internal)		set_cnt_src ()=1 (external)		Interrupt?
	Get_command()	get_position()	get_command()	get_position()	
Case 1	Doesn't change	Remain 0	Doesn't change	Stop at a deceleration position	Home Int
Case 2-1	Doesn't change	Remain 0	Doesn't change	Stop at a deceleration position	Home Int
Case 2-2	Doesn't change	Stop at a EL position	Doesn't change	Stop at a EL position	EL Int
Case 3	Doesn't change	Stop at a EL position	Doesn't change	Stop at a EL position	EL Int

- In Case 1, the axis will be stopped immediately when axis detected the edge of ORG signal but it might stop at anywhere within the range of ORG signal that means the home position is inaccurate after home move was executed many times.

- The feedback counter of PCI-8134 will be reset to zero while the motion is hitting the edge of ORG signal.

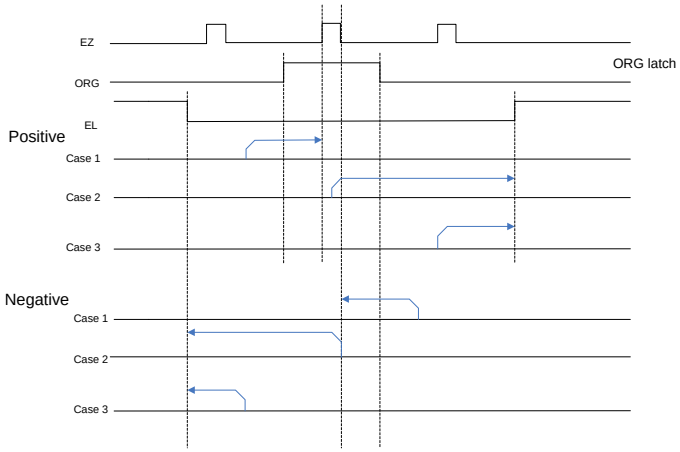
- As Do Not Latch mode, the axis will start searching EZ signal after the ORG signal was detected within 5 clock periods.

- In Case 2-2 & 3, the axis will hit and then stop at the edge of EL signal anyway because the edge of EZ signal locates behind the start point of Home Move that the axis won't detect the edge of EZ signal anymore.

- In Case 2-1, if there are two or more EZ signals in the system, the axis will search next EZ signal because the ORG signal was turned ON continuously.

- After normal home finished like case 1, users have to copy to position value to command counter and target position counter at the same time.

PCI-8134 Home Mode 1 + ORG latch



PCI-8134 Home Mode 1 + ORG latch @ 8134.DLL/8134A.DLL

While the motion hits the edge of EZ or EL				
	set_cnt_src()=0 (internal)		set_cnt_src ()=1 (external)	
	get_command()	get_position()	get_command()	get_position()
Case 1	Doesn't change	Reset to 0	Doesn't change	Reset to 0
Case 2	Doesn't change	Stop at a EL position	Doesn't change	Stop at a EL position
Case 3	Doesn't change	Stop at a EL position	Doesn't change	Stop at a EL position

Counter status after Home Move Completed (Motion Done)					
	Set_cnt_src()=0 (internal)		set_cnt_src ()=1 (external)		Interrupt?
	Get_command( )	get_position()	get_command()	get_position()	
Case 1	Doesn't change	Remain 0	Doesn't change	Stop at a deceleration position	Home Int
Case 2	Doesn't change	Stop at a EL position	Doesn't change	Stop at a EL position	EL Int
Case 3	Doesn't change	Stop at a EL position	Doesn't change	Stop at a EL position	EL Int

- In Case 1, the axis will be stopped immediately when axis detected the edge of ORG signal but it might stop at anywhere within the range of ORG signal that means the home position is inaccurate after home move was executed many times.

- The feedback counter of PCI-8134 will be reset to zero while the motion is hitting the edge of ORG signal.

- As Do Not Latch mode, the axis will start searching EZ signal after the ORG signal was detected within 5 clock periods.

- In Case 2 & 3, the axis will hit and then stop at the edge of EL signal anyway because the edge of EZ signal locates behind the start point of Home Move that the axis won't detect the edge of EZ signal anymore.

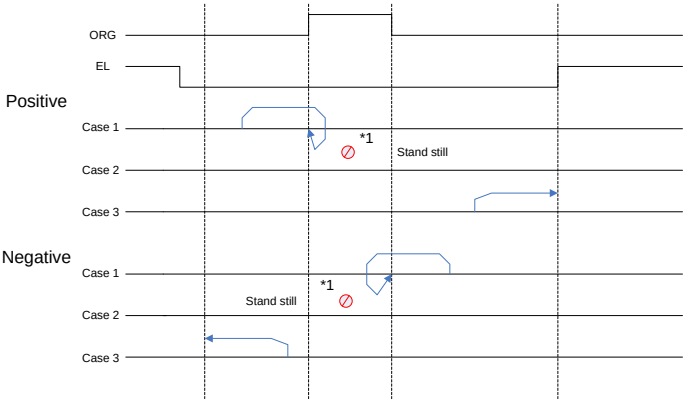
- In Case 2 & 3, if there are two or more EZ signals in the system, the axis never stop if the axis starts from first EZ signal because there is no ORG signal was triggered prior to EZ searching.

- After normal home finished like case 1, users have to copy to position value to command counter and target position counter at the same time.

PCI-8134A Home Mode 0 & Home Mode 1

The timing charts of Home Mode 0 and 1 of PCI-8134A follow.

PCI-8134A Home Mode 0



*1 In the case 2 of PCI-8134A Home Mode 0, The axis will stand still and reset counter to 0 But won' t issue home interrupt after user issued home move command.

PCI-8134A Home Mode 0 @ 8134.DLL/8134A.DLL/8134A.DLL

While the motion hits the edge of ORG or EL				
	Set_cnt_src()=0 (internal)		set_cnt_src ()=1 (external)	
	Get_command( )	get_position()	get_command()	get_position()
Case 1	Doesn't change	Reset to 0	Doesn't change	Reset to 0
Case 2	Doesn't change	Stand Still	Doesn't change	Stand Still
Case 3	Doesn't change	Stop at a EL position	Doesn't change	Stop at a EL position

Counter status after Home Move Completed (Motion Done)					
	set_cnt_src()=0 (internal)		set_cnt_src ()=1 (external)		Interrupt?
	get_command()	get_position()	get_command()	get_position ()	
Case 1	**Doesn't change**	**Remain 0**	**Doesn't change**	**Remain 0**	**Home Int**
Case 2	**Doesn't change**	**Reset to 0**	**Doesn't change**	**Reset to 0**	**No Int**
Case 3	**Doesn't change**	**Stop at a EL position**	**Doesn't change**	**Stop at a EL position**	**EL Int**

- Home point is at the first edge of ORG signal when home move executing. At left or right side of edge depends on home move direction.

- If axis is not at ORG, the axis will search the edge of ORG as home point.

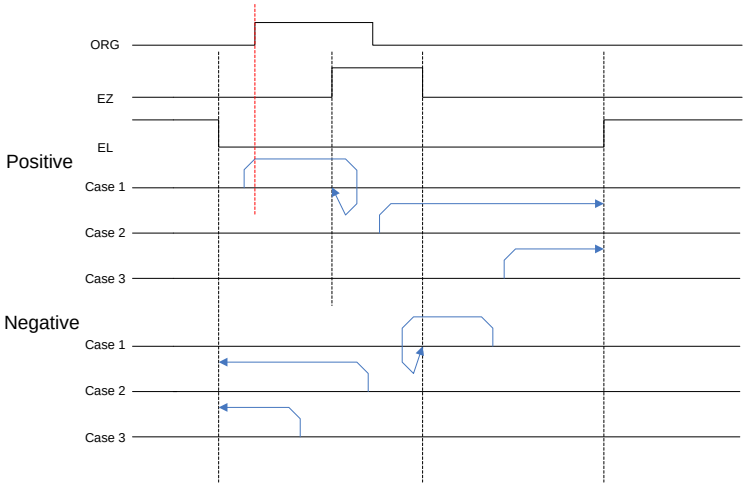
- In Case 1, the axis will slow down and reverse to search the ORG edge again and then stop at the edge of ORG signal precisely.

- The feedback counter of PCI-8134A will be reset to zero while the motion is hitting the edge of ORG signal.

- In Case 2, the axis will be standstill and reset counter to 0 but won't issue home interrupt.

- After normal home finished like case 1, users have to copy to position value to command counter and target position counter at the same time.

PCI-8134A Home Mode 1



PCI-8134A Home Mode 1 @ 8134.DLL/8134A.DLL

DLL version: 110420, Driver version: 101109

While the motion hits the edge of EZ or EL				
	set_cnt_src()=0 (internal)		**set_cnt_src ()=1 (external)**	
	get_command()	**get_position()**	**get_command()**	**get_position()**
Case 1	Doesn't change	Reset to 0	Doesn't change	Reset to 0
Case 2	Doesn't change	Stop at a EL position	Doesn't change	Stop at a EL position
Case 3	Doesn't change	Stop at a EL position	Doesn't change	Stop at a EL position

Counter status after Home Move Completed (Motion Done)					
	set_cnt_src()=0 (internal)		**set_cnt_src ()=1 (external)**		**Interrupt?**
	get_command()	**get_position()**	**get_command()**	**get_position()**	
Case 1	**Doesn't change**	**Remain 0**	**Doesn't change**	**Remain 0**	Home Int
Case 2	**Doesn't change**	**Stop at a EL position**	**Doesn't change**	**Stop at a EL position**	EL Int
Case 3	**Doesn't change**	**Stop at a EL position**	**Doesn't change**	**Stop at a EL position**	EL Int

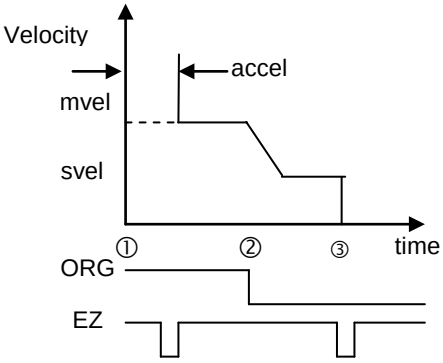
- In Case 1, the axis will search the edge of EZ signal after the ORG signal and then stop at EZ edge precisely.

- The feedback counter of PCI-8134 will be reset to zero while the motion is hitting the edge of EZ or EL signal.

- In Case 2 & 3, the axis will hit and then stop at the edge of EL signal anyway because the edge of EZ signal locates behind the start point of Home Move that the axis won't detect the edge of EZ signal anymore.

- After normal home finished like case 1, users have to copy to position value to command counter and target position counter at the same time.

(3) Home_mode=2: both ORG and index signal are useful. The ORG signal lets the PCI-8134/PCI-8134A decelerate to starting velocity and then EZ signal stops OUT and DIR pins from outputting pulses to complete the home return.



Note: If the starting velocity is zero, the axis will work properly in home mode 2.

Relative Function:

set_home_config(), home_move(), v_stop(): Refer to section 6.11

4.1.7 Manual Pulser Mode

For manual operation of a device, you may use a manual pulser such as a rotary encoder. The PCI-8134/PCI-8134A can input signals from the pulser and output corresponding pulses from the OUT and DIR pins, thereby allowing you to simplify the external circuit and control the present position of axis. This mode is effective between a **manu_move()** command is written and a **v_stop()** command.

The PCI-8134/PCI-8134A receives plus and minus pulses (CW/CCW) or 90 degrees phase difference signals(AB phase) from the pulser at PA and PB pins. The 90°phase difference signals can be input through multiplication by 1, 2 or 4. If the AB pahse input mode is selected, the PA and PB signals should be with 90° phase shifted, and the position counting is increasing when the PA signal is leasding the PB signal by 90° phase.

Also, one pulser may be used for 'X' and 'Y' axes while internally distributing the signals appropriately to two axes. To set the input signal modes of pulser, use **set_manu_iptmode()** function. Then write **manu_move()** to begin manual operation function. User must write **v_stop()** command in order to end this function and begins to operate at another mode.

The error input of PA and PB can be used to generate IRQ. The following two situations will be considered as error input of PA and PB signals. (1) The PA and PB signals are changing simultaneously. (2) The input pulser frequency is higher than the maximum output frequency 2.4M pps. Set bit 14 of INT factor will enable the IRQ when error happen.

Maximum moving velocity in this mode can be limited by setting max_vel parameter in **manu_move()** function.

Relative Function:

set_manu_iptmode(), manu_move(), v_stop(): Refer to section 6.12

4.2 Motor Drive Interface

The PCI-8134/PCI-8134A provides the INP, ERC and ALM signals for servomotor driver's control interface. The INP and ALM are used for feedback the servo driver's status. The ERC is used to reset the servo driver's deviation counter under special conditions.

4.2.1 INP

Usually, servomotor driver with pulse train input has a deviation (position error) counter to detect the deviation between the input pulse command and feedback counter. The driver controls the motion of servomotor to minimize the deviation until it becomes 0. Theoretically, the servomotor operates with some time delay from command pulses. Accordingly, when the pulse generator stops outputting pulses, the servomotor does not stop but keep running until the deviation counter become zero. At this moment, the servo driver sends out the in-position signal (INP) to the pulse generator to indicate the motor stops running.

Usually, the PCI-8134/PCI-8134A stops outputting pulses upon completion of outputting designated pulses. But by setting ***inp_enable*** parameter in ***set_inp_logic()*** function, you can delay the completion of operation to the time when the INP signal is turned on. Status of **motion_done()** and INT signal are also delayed. That is, when performing under position control mode, the completion of **start_a_move()**, **start_r_move()**, **start_s_move()**… functions are delayed until INP signal is turned ON.

However, EL or ALM signal or the completion of home return does not cause the INP signal to delay the timing of completion. The INP signal may be a pulse signal, of which the shortest width is 5 micro seconds.

The in-position function can be enable or disable. The input logic polarity isalso programmable by software function:**set_inp_logic()**. The signal status can be monitored by software function: **get_io_status()**.

4.2.2 ALM

The ALM pin receives the alarm signal output from the servo driver. The signal immediately stops the PCI-8134/PCI-8134A from generating pulses or stops it after deceleration. If the ALM signal is in the ON status at the start, the PCI-8134/PCI-8134A outputs the INT signal without generating

any command pulse. The ALM signal may be a pulse signal, of which the shortest width is a time length of 5 micro seconds.

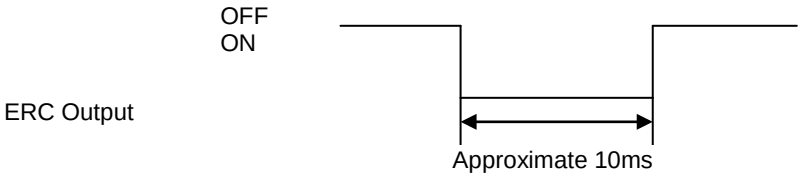
You can change the input logic by ***set_alm_logic()*** function. Whether or not the PCI-8134/PCI-8134A is generating pulses, the ALM signal lets it output the INT signal.. The ALM status can be monitored by software function: ***get_io_status()***. The ALM signal can generate IRQ by setting the bit 2 of INT. factor in software function: **set_int_factor().**

4.2.3 ERC

The deviation counter clear signal is inserted in the following 4 situations:

(1) home return is complete;
(2) the end-limit switch is active;
(3) an alarm signal stops OUT and DIR signals;
(4) an emergency stop command is issued by software operator.

Since the servomotor operates with some delay from pulse generated from the PCI-8134/PCI-8134A, it keeps operating by responding to the position error remaining in the deviation counter of the driver if the ±EL signal or the completion of home return stops the PCL5023 from outputting pulses. The ERC signal allows you to immediately stop the servomotor by resetting the deviation counter to zero. The ERC signal is output as an one-shot signal. The pulsewidth is a time length of 10ms. The ERC signal will automatically output when ±EL signals, ALM signal is turned on to immediately stop the servomotor. User can set the ERC pin output enable/disable by ***set_erc_enable()*** function. ERC pin output is set output enabled when initializing.



CAUTION

Due to differences between the motion chipsets on the PCI-8134 and PCI-8134A, ERC output pulse width with the PCI-8134A may be less than originally output by the PCI-8134.

4.3 The Limit Switch Interface and I/O Status

In this section, the following I/O signals' operations are described.

- SD: Ramping Down sensor
- ±EL: End-limit sensor
- ORG: Origin position
- SVON and RDY

I/O status readback

In any operation mode, if an ±EL signal is active during moving condition, it will cause PCI-8134/PCI-8134A to stop output pulses automatically. If an SD signal is active during moving condition, it will cause PCI-8134/PCI-8134A to decelerate.

4.3.1 SD

The ramping-down signals are used to slow-down the control output signals (OUT and DIR) when it is active. The signals are very useful to protect the mechanism moving under high speed toward the mechanism limit. PSD indicates ramping-sown signal in plus (+) direction and MSD indicates ramping-down signal in minus (-) direction.

During varied speed operation in the home return mode or continuous operation mode, the ramping-down signal in the moving direction lets the output control signals (OUT and DIR) ramp down to the pre-setting starting velocity.

The ramping-down function can be enable or disable by software function: **set_sd_logic()**. The input logic polarity, level operation mode, or latched input mode can also be set by this function. The signals status can be monitored by **get_io_status()**.

4.3.2 EL

The end-limit signals are used to stop the control output signals (OUT and DIR) when the end-limit is active. PEL signal indicates end-limit in positive (plus) direction. MEL signal indicates end-limit in negative (minus) direction. When the output pulse signals (OUT and DIR) are toward positive direction, the pulse train will be immediately stopped when the PEL signal is inserted, while the MEL signal is meaningless in this case, and vise versa. When the PEL is inserted and the output pulse is fully stop, only the negative (minus) direction output pulse can be generated for moving the motor to negative (minus) direction.

The end-limit signals can be used to generate the IRQ by setting the bit 0 of INT. factor in software function: **set_int_factor()**.

You can use either 'a' contact switch or 'b' contact switch by setting the dip switch S1. The PCI-8134/PCI-8134A is delivered from the factory with all bits of S1 set to OFF.

The signal status can be monitored by software function: **get_io_status().**

4.3.3 ORG

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

There are three home return modes, you can select one of them by setting "*home_mode*" argument in software function: set_home_config(). Note that if home_mode=1 or 2, the ORG signal must be ON or latched during the EZ signal is inserted (EZ=0). The logic polarity of the ORG signal, level input or latched input mode are selectable by software function: **set_home_config()**.

After setting the configuration of home return mode by **set_home_config()**, a home_move() command can perform the home return function.

The ORG signal can also generate IRQ signal by setting the bit 5 of interrupt reason register (or INT. factor) in software function: **set_int_factor()**.

4.3.4 SVON and RDY

The SVON signals are controlled by software function: **_8134_Set_SVON()**. The function set the logic of SVON signal. The signal status of SVON pins can be monitored by software function: **get_io_status()**.

RDY pins are dedicated for digital input use The status of this signal can be monitored by software function get_io_status(). The RDY signal can also generate IRQ signal by setting the bit 23 of INT. factor in software function: set_int_factor(). Note that interrupt is generated when RDY signal from high to low.

The PCI-8134A supports neither RDY signal connection nor interrupt function.

4.4 The Encoder Feedback Signals (EA, EB, EZ)

The PCI-8134/PCI-8134A has a 28-bits binary up/down counter for managing the present position for each axis. The counter counts signals input from EA and EB pins.

It can accept 2 kinds of pulse input: (1). plus and minus pulses input(CW/CCW mode); (2). 90° phase difference signals (AB phase mode). 90° phase difference signals may be selected to be multiplied by a factor of 1,2 or 4. 4x AB phase mode is the most commonly used for incremental encoder input. 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 depends on its turning direction. These input modes can be selected by ***set_pls_iptmode()*** function.

To enable the counters counting pulses input from (EA, EB) pins, set "***cnt_src***" parameter of software function ***set_cnt_src()*** to 1.

Plus and Minus Pulses Input Mode(CW/CCW Mode)

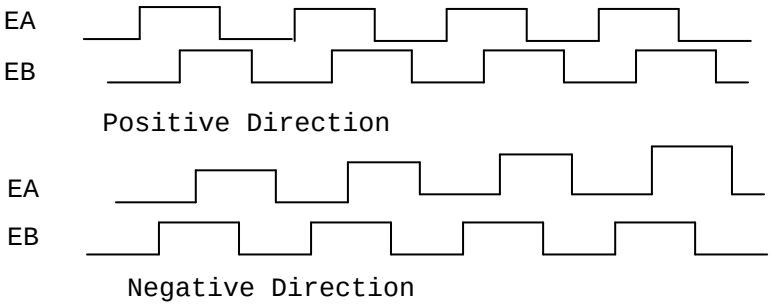
The pattern of pulses in this mode is the same as ***Dual Pulse Output Mode*** in Pulse Command Output section, expect that the input pins are EA and EB.

In this mode, pulse from EA causes the counter to count up, whereas EB caused the counter to count down.

90° phase difference signals Input Mode(AB phase Mode)

In this mode, the EA signal is 90° phase leading or lagging in comparison with EB signal. Where "lead" or "lag' of phase difference between two signals is caused by the turning direction of motors. The up/down counter counts up when the phase of EA signal leads the phase of EB signal.

The following diagram shows the waveform.



The encoder error interrupt is provided to detect abnormal situation. Simultaneously changing of EA and EB signals will cause an encoder error. If bit #14 of the interrupt factor register (INT factor) is set as 1, the IRQ will be generated when detect encoder error during operation.

The index inputs (EZ) signals of the encoders are used as the "ZERO" index. This signal is common on most of the rotational motors. EZ can be used to define the absolute position of the mechanism. The input logic polarity of the EZ signals is programmable by software function ***set_home_config().*** The EZ signals status of the four axis can be monitored by ***get_io_status()***.

Relative Function:

set_cnt_src(), set_pls_iptmode(): Refer to section 6.4

4.5 Multiple PCI-8134/PCI-8134A Cards Operation

The software function library support maximum up to 12 PCI-8134/PCI-8134A Cards that means maximum up to 48 axes of motors can be controlled. Since PCI-8134/PCI-8134A has the characteristic of Plug-and-Play, users do not have to care about setting the Based address and IRQ level of cards. They are automatically assigned by the BIOS of system when booting up. Users can utilize Motion Creator to check if the plugged PCI-8134/PCI-8134A cards are successfully installed and see the Base address and IRQ level assigned by BIOS.

One thing needed to be noticed by users is to identify the card number of PCI-8134/PCI-8134A when multiple cards are applied. The card number of one PCI-8134/PCI-8134A depends on the locations on the PCI slots. They

are numbered either from left to right or right to left on the PCI slots. These card numbers will affect the corresponding axis number on the cards. And the axis number is the first argument for most functions called in the library. So it is important to identify the axis number before writing application programs. For example, if 3 PCI-8134/PCI-8134A cards are plugged in the PCI slots. Then the corresponding axis number on each card will be:

Axis No. / Card No.	Axis 1	Axis 2	Axis 3	Axis 4
1	0	1	2	3
2	4	5	6	7
3	8	9	10	11

If we want to accelerate Axis 3 of Card2 from 0 to 10000pps in 0.5sec for Constant Velocity Mode operation. The axis number should be 6. The code on the program will be:

```
v_move(6, 0, 10000, 0.5);
```

To determine the right card number, Try and Error may be necessary before application. Motion Creator can be utilized to minimize the search time.

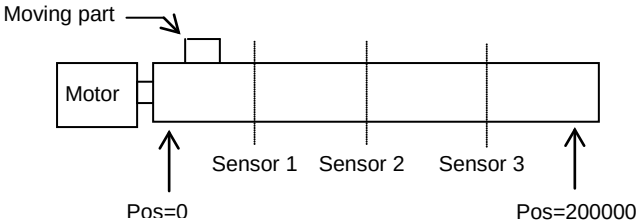
The newest DLL supports the combination of both PCI-8134 and PCI-8134A in one system.

4.6 Change Speed on the Fly

You can change the velocity profile of command pulse ouput during operation by ***v_change()*** function. This function changes the maximum velocity setting during operation. However, if you operate under “Preset Mode” (like start_a_move(),…), you are not allowed to change the acceleration parameter during operation because the deceleration point is pre-determined. But changing the acceleration parameter when operating under “Constant Velocity Mode” is valid. Changing speed pattern on the fly is valid no matter what you choose “Trapezoidal Velocity Profile” or “S-curve Velocity Profile”. Here we use an example of Trapezoidal velocity profile to illustarte this function.

Example: There are 3 speed change sensor during an absolute move for 200000 pulses. Initial maximum speed is 10000pps. Change to 25000pps if

Sensor 1 is touched. Change to 50000pps if Sensor 2 is touched. Change to 100000pps if Sensor 3 is touched. Then the code for this application and the resulting velocity profiles are shown below.



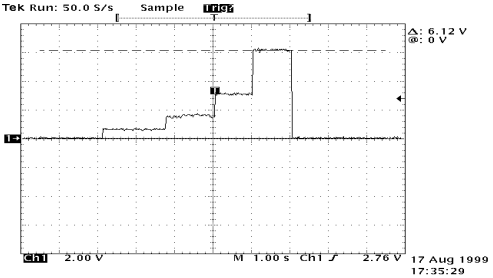

```
#include “pci_8134.h”

start_a_move(axis, 200000.0, 1000, 10000, 0.02);

while(!motion_done(axis))

  {
                                  // Get Sensor’s information from other I/O card
      if((Sensor1==High) && (Sensor2==Low) && (Sensor3 == Low))
                  v_change(axis, 25000, 0.02);
      else if((Sensor1==Low) && (Sensor2==High) && (Sensor3 == Low))
                  v_change(axis, 50000, 0.02);
      else if((Sensor1==Low) && (Sensor2==Low) && (Sensor3 == High))
                  v_change(axis, 100000, 0.02);
  }
```

Where the informations of three sensors are acquired from other I/O card. And the resulting velocity profile from experiment is shown below.



Relative Function:

v_change(): Refer to section 6.5

4.7 Interrupt Control

The PCI-8134/PCI-8134A motion controller can generate INT signal to host PC according to 13 types of factors, refer to **set_int_factor()** function for more details.. The INT signal is output when one or more interrupt factors occur on either axis. To judge on which axis the interrupt factors occur, use **get_int_axis()** function. The interrupt status is not closed until **get_int_status()** function is performed.

5

Motion Creator

After installing all the hardware properly according to Chapter 2, 3, configuring cards and checkout are required before running. This chapter gives guidelines for establishing a control system and manually exercising the PCI-8134/PCI-8134A cards to verify correct operation. Motion Creator provides a simple yet powerful means to setup, configure, test and debug motion control system that uses PCI-8134/PCI-8134A cards.

Note that Motion Creator is available only for Windows XP/7 with the screen resolution higher than 800x600 environment and can not run on DOS system.

5.1 Main Menu

Main Menu will appear when executing Motion Creator. Figure 5.1 shows the Main Menu.

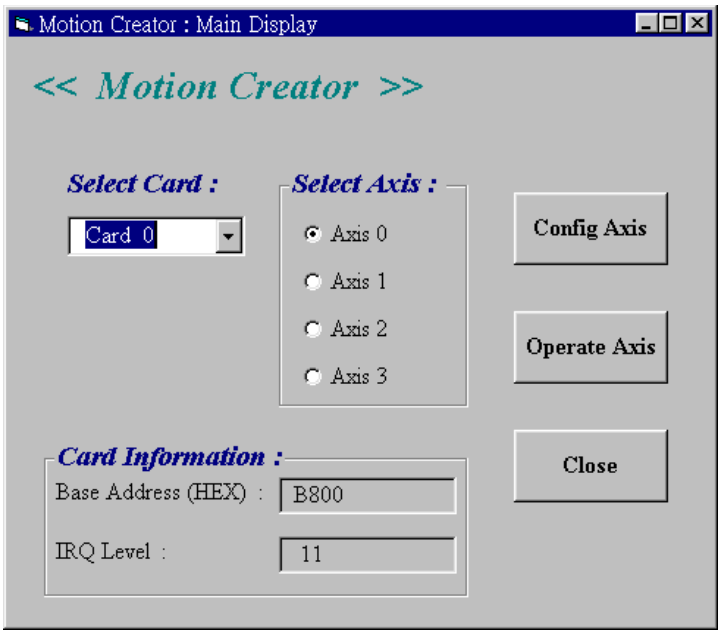
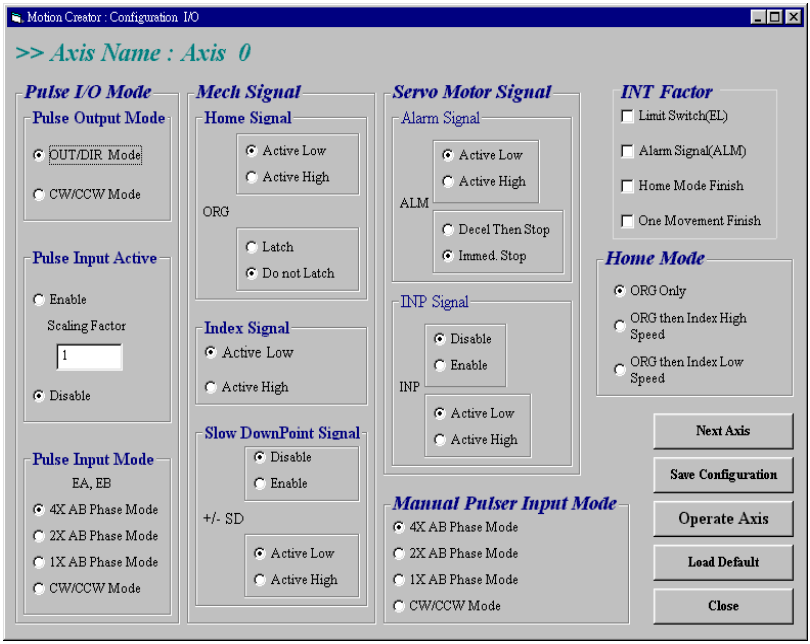


Figure 5.1 Main Menu of Motion Creator

From main menu window all PCI-8134/PCI-8134A cards and their axes and the corresponding status can be viewed. First of all, check if all the PCI-8134/PCI-8134A cards which are plugged in the PCI-Bus can be viewed on "Select Card" column. Next select the card and axis you want to configure and operate. Since there are totally four axes on a card, the axis number of first axis on n-the card will be numbered as 4*(n-1). Base address and IRQ level of the card are also shown on this window.

5.2 Axis Configuration Window

Press the “Config Axis” button on the Main Menu will enter the Axis



Configuration window. Figure 5.2 shows the window.

Figure 5.2 Axis Configuration Window

the Axis Configuration window includes the following setting items which cover most I/O signals of PCI-8134/PCI-8134A cards and part of the interrupt factors.

- Pulse I/O Mode:

  Related functions:

  * set_pls_outmode() for “Pulse Output Mode” property.
  * set_cnt_src() for “Pulse Input Active” property.

* set_pls_iptmode() for “Pulse Input Mode” property.

- Mechanical Signal:

  Related functions:

  - set_home_config() for “Home Signal” and “Index Signal” property.
  - set_sd_logic() for “Slow Down Point Signal” property.
- Servo Motor Signal:

  Related functions:

  - set_alm_logic() for “Alarm Signal” property.
  - set_inp_logic() for “INP” property.
- Manual Pulser Input Mode:

  Related functions:

  - set_manu_iptmode() for “Manual Pulser Input Mode” property.
- Interrupt Factor:

  Related functions:

  - set_int_factor() for “INT Factor” property.
- Home Mode:

  Related functions:

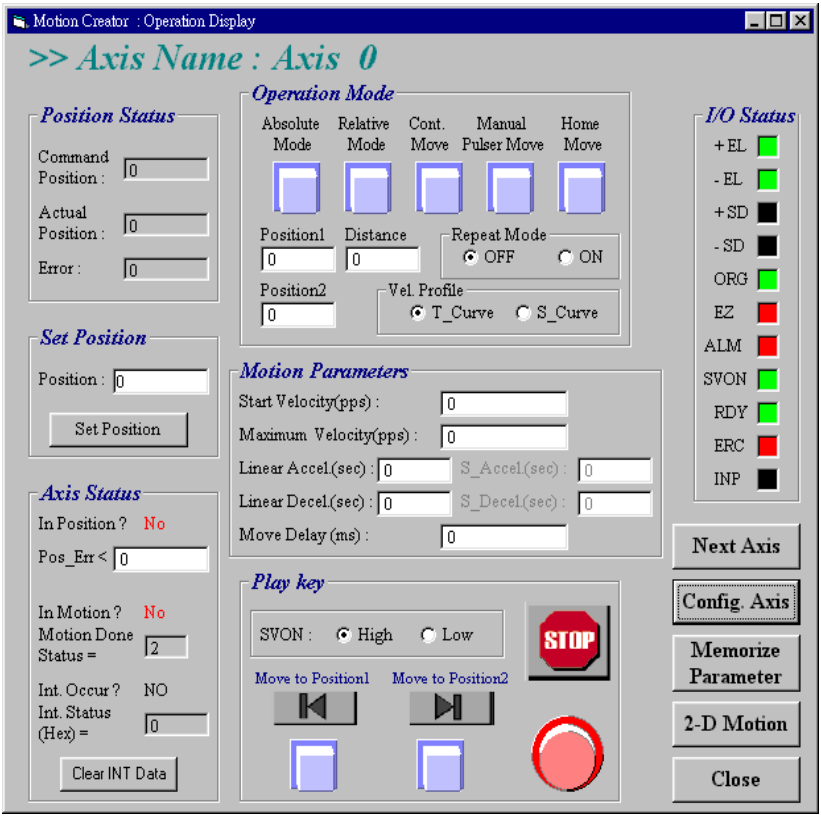
  - set_home_config() for “Home Mode” property.

The details of each section are shown at its related functions.

After selecting all the items you want to configure, user can choose to push the “Save Configurations “ button on the right bottom side. If you push this button, all the configurations you select for system integration will be saved to a file called “8134.cfg”. This file is very helpful when user is developing their own application programs. The following example illustrate how to make use of this function. This example program is shown in C language form.

```
Main ()
{
    _8134_intial ();                    // Initialize the PCI-8134/PCI-8134A cards
    _8134_Set_Config ();  // Configure PCI-8134/PCI-8134A cards according
    :
                                        //to 8134.cfg
    :
}
```

Where *_8134_initial ()* and *_8134_Set_Config ()* can be called from the function library we provide. *_8134_initial ()* should be the first function called within main {} function. It will check all the PCI-8134/PCI-8134A existed and give the card a base address and IRQ level. *_8134_Set_Config ()* will configure the PCI-8134/PCI-8134A cards according to "8134.cfg". That is, the contents of Axis Configuration Window can be transferred to the



application program by this function called.

Figure 5.3 Axis Operation window

5.3 Axis Operation Windows

Press the “Operate Axis” button on the Main Menu or Axis Configuration Menu will enter the Axis Configuration window. Figure 5.3 shows the window. User can use this window to command motion, monitor all the I/O status for the selected axis. This window includes the following displays and controls:

- Motion Status Display,
- Axis Status Display
- I/O Status Display
- Set Position Control
- Operation Mode Control
- Motion Parameter Control
- Play Key Control
- Velocity Profile Selection
- Repeat Mode

5.3.1 Motion Status Display

The Motion Status display provides a real-time display of the axis’s position in the Command, Actual, Error fields. Motion Creator automatically updates these command, actual and error displays whenever any of the values change.

When Pulse Input Active property is Axis Configuration Window is set to Enable, the Actual Position read will be from the external encoder inputs (EA, EB). Else, it will display the command pulse output when set to Disable.

5.3.2 Axis Status Display

The Axis Status display provides a real-time display of the axis’s status.

It displays the status (Yes (for logical True) or No (for logical False)) for In Position or In Motion or displays there is Interrupt Events Occurs. When In motion, you can check the motion done status in the next column. In Position range can be specified in the Pos_Err column.

5.3.3 I/O Status Display

Use I/O Status display to monitor the all the I/O status of PCI-8134/PCI-8134A. The Green Light represents ON status, Red Light represents OFF status and BLACK LIGHT represents that I/O function is disabled. The

ON/OFF status is read based on the setting logic in Axis Configuration window.

5.3.4 Set Position Control

Use the Set Position Control to arbitrarily change the actual position of axis.Write the position wanting to specify into the column and click the “Set Position” button will set the actual position to the specified position.

5.3.5 Operation Mode Control

There are four Operation Modes mentioned in Chapter 4 can be tested in the Axis Operation window. They are “Continuous Move Mode”, “Preset Mode Operation”, “Home Mode Operation”, “Manual Mode Operation”.

- Continuous Move Mode:

Press “Continuous Move” button will enable Continuous Velocity motion as specified by values entered in “Start Velocity” and “Maximum Velocity” 2 fields of Motion Parameters Control. The steady state moving velocity will be as specified by “Maximum Velocity”. Press → to move forward or ← to move backward. Press “STOP” to stop moving.

- Preset Mode:

Press “Absolute Mode” to enable absolute motion as specified by values entered in “Position 1” and “Position 2” 2 fields. When selected, “Distance” field for “Relative Mode” is disabled. Press → to move to Position 2 or ← to move to Position 1. Press “STOP” to stop motion.

Also, user can specify repetitive motion in “Absolute Mode” by setting “Repeat Mode” to “ON” state. When “Repeat Mode” goes “ON” and either → or ← is pressed., axis starts repetitive motion between Position 1 and Position 2 until “Repeat Mode” goes “OFF” as “STOP” are clicked.

Press “Relative Mode” to enable relative motion as specified by values entered in “Distance” fields. When selected, “Position 1” and “Position 2” fields for “Absolute Mode” is disabled. Press → to move forward to a distance relative to present position as specified by “Distance” or ← to move backward.

Note that both “Absolute Mode” and “Relative Mode” are operated under a trapezoidal velocity profile as specified by Motion Parameters Control.

- Home Return Mode:

Press “Home Move” button will enable Home Return motion. The home returning velocity is specified by settings in Motion Parameters Control. The arriving condition for Home Return Mode is specified in Axis Configuration

Window. Press → to begin returning home function. Press “STOP” to stop moving.

- Manual Pulser Mode:

Press “Manual Pulser Move” button will enable motion controlled by hand wheel pulser. Using this function, user can manually operate the axis thus verify operation. The maximum moving velocity is limited as specified by “Maximum Velocity”. Press “STOP” to end this mode.

Do remember to press “STOP” to end operation under this mode. Otherwise, operations under other modes will be inhibited.

5.3.6 Motion Parameters Control

Use the Motion Parameters with the Operation Mode Control to command motion.

- Starting Velocity: Specify the starting moving speed in pulses per second.
- Maximum Velocity: Specify the maximum moving speed in pulses per second.
- Acceleration: Specify the acceleration in pulses per second square.
- Move delay: Specify time in mini seconds between movement.
- S curve Acc/dec Time: Specify time in mini second for S_curve Movement.

5.3.7 Play Key Control

Use buttons in Play Key Control to begin or end operation.

: click button under this symbol to begin moving to Positions 2 in Absolute Mode or moving forward in other modes.

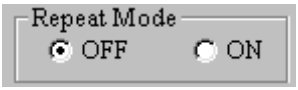
: click button under this symbol to begin moving to Positions 1 in Absolute Mode or moving backward in other modes.

: click button under this symbol to stop motion under any mode. Note that this button is always in latch mode. Click again to release “STOP” function.

5.3.8 Velocity Profile Selection

: Click T_Curve or S_curve to select preset movement velocity profile. The relative parameter settings are in Motion Parameter Frame.

5.3.9 Repeat Mode



: Repeat mode is only for absolute and relative mode. After choosing a operation mode and click repeat mode on, you can press play key to make axis run between position 1 and 2 (in absolute mode) or run between a range (relative mode). It is useful on demonstrations. Use Stop button to stop this operation.

6

Function Library (8134.DLL)

This chapter describes the supporting software for PCI-8134/PCI-8134A cards. User can use these functions to develop application program in C or Visual Basic or C++ language.

6.1 List of Functions

Initialization		**Section 6.3**
W_8134_Initial	Card initialization	
W_8134_InitialA	Card initialization type A	
W_8134_Close	Card Close	
W_8134_Set_Config	Configure card according to Motion Creator's setting	
W_8134_Get_IRQ_Channel	Get IRQ channel	
W_8134_Get_Base_Addr	Get Base Address	

Pulse Input/Output Configuration		**Section 6.4**
set_pls_outmode	Set pulse command output mode	
set_pls_iptmode	Set encoder input mode	
set_cnt_src		

Continuously Motion Mode		**Section 6.5**
v_move	Accelerate an axis to a constant velocity with trapezoidal profile	
sv_move	Accelerate an axis to a constant velocity with S- curve profile	
v_change	Change speed on the fly	

v_stop	Decelerate to stop
set_sd_stop_mode	Set slow down stop mode
fix_max_speed	Fix maximum speed setting
unfix_max_speed	Unfix maximum speed setting
get_current_speed	Get current speed in pps
verify_speed	Get the minimum acceleration time under the speed setting

Trapezoidal Motion Mode	**Section 6.6**
a_move	Perform an absolute trapezoidal profile move and wait for finish
start_a_move	Start an absolute trapezidal profile move
r_move	Perform a relative trapezoidal profile move and wait for finish
start_r_move	Start a relative trapezoidal profile move
t_move	Perform a relative non-symmetrical trapezoidal profile move and wait for finish
start_t_move	Start a relative non-symmetrical trapezidal profile move
start_ta_move	Start an absolute non-symmetrical trapezidal profile move
ta_move	Start an absolute non-symmetrical trapezoidal profile move and wait for finish
wait_for_done	Wait for an axis to finish
set_rdp_mode	Set Ramping down mode

S-Curve Profile Motion	**Section 6.7**
s_move	Perform an absolute S-curve profile move and wait for finish
start_s_move	Start an absolute S-curve profile move
rs_move	Perform a relative S-curve profile move and wait for finish
start_rs_move	Start a relative S-curve profile move
tas_move	Perform an absolute non-symmetrical S-curve profile move and wait for finish
start_tas_move	Start an absolute non-symmetrical S-curve profile move

Multiple Axes Point to Point Motion	**Section 6.8**
start_move_all	Start a multi-axis trapezodial profile move

move_all	Perform a multi-axis trapezodial profile move and wait for finish

start_sa_move_all Start a multi-axis absolute S-curve profile move wait_for_all Wait for all axes to finish

Linear Interpolated Motion	**Section 6.9**
move_xy	Perform a 2-axis linear interpolated move for X & Y and wait for finish
move_zu	Perform a 2-axis linear interpolated move for Z & U and wait for finish
start_move_xy	Start a 2-axis linear interpolated move for X & Y
start_move_zu	Start a 2-axis linear interpolated move for Z & U

Interpolation Parameters Configuring	**Section 6.10**
map_axes	Maps coordinated motion axes x, y, z…
set_move_speed	Set the vector velocity
set_move_accel time	Set the vector acceleration
set_move_ratio	Set the axis resolution ratios

Home Return Mode	**Section 6.11**
set_home_config	Set or get the home/index logic configuration
home_move	Start a home return actionulse command output mode

Manual Pulser Motion		Section 6.12
set_manu_iptmode	Set pulser input mode and operation mode	
manu_move	Begin a manual pulser movement	

Motion Status		Section 6.13
motion_done	Check if the axis is in motion	

Servo Drive Interface		Section 6.14
set_alm_logic	Set alarm logic and alarm mode	
set_inp_logic	Set In-Position logic and enable/disable	
set_sd_logic	Set slow down point logic and enable/disable	
set_erc_enable	Set the ERC output enable/disable	

I/O Control and Monitoring		Section 6.15
W_8134_Set_SVON	Set the state of general purpose output	
bit get_io_status	Get all the I/O staus of PCI-8134	

Position Control	Section 6.16

Interrupt Control		Section 6.17
W_8134_INT_Enable	Set Interrupt Event enable	
W_8134_INT_Disable	Set Interrupt Event enable	
W_8134_Set_INT_Control set_int_factor	get_int_statusg factors	
Get the interrupting status of axis		
link_axis_interrupt	Link a callback function for interrupt	

6.2 C/C++ Programming Library

This section gives the details of all the functions. The function prototypes and some common data types are decelerated in **PCI-8134.H**. These data types are used by PCI-8134/PCI-8134A library. We suggest you to use these data types in your application programs. The following table shows the data type names and their range.

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

6.3 Initialization

@ Name

W_8134_Initial – Card Initialization
W_8134_InitialA – Another Card* Initialization *Method
W_8134_Close – Card Close
W_8134_Set_Config – Configure Card according to Motion Creator
W_8134_Get_IRQ_Channel – Get the card's IRQ number
W_8134_Get_ Base_Addr – Get the card's base address

@ Description

W_8134_Initial:

This function is used to initialize PCI-8134 card. It has to be initialized by this function before calling other functions. The parameter definitions of this function are different from OS. Please pay attention it.

W_8134_InitialA:

This function is like above one. The only difference is parameter definition. We suggest that users use this function for card initialization because this function is OS independent.

W_8134_Close:

This function must be called before the program ends.

W_8134_Set_Config:

This function is used to configure PCI-8134 card. All the I/O configurations and some operating modes appeared on "Axis Configuration Window" of Motion Creator will be set to PCI-8134. Click "Save Configuration" button on the "Axis Configuration Window" if you want to use this function in the application program. Click "Save Configuration" button will save all the configurations to a file call "*8134.cfg*". This file will appear in the Windows' system directory.

W_8134_Get_IRQ_Channel :

This function is used to get the PCI-8134 card's IRQ number.

W_8134_Get_Base_Addr :

This function is used to get the PCI-8134 card's base address.

@ *Syntax*

C/C++ (DOS)

```
U16 _8134_Initial (U16 *existCards, PCI_INFO *info)
U16 _8134_Close(U16 cardNo)
U16 _8134_Set_Config(char* filename)
```

C/C++ (Windows)

```
U16 W_8134_Initial(U16 *existCards, PCI_INFO *pciInfo)
    (Windows 9x Only)
U16 W_8134_Initial(I32 cardNo) (Windows NT/2K/XP)
U16 W_8134_InitialA(I16 *Totalcard) (All Windows)
U16 W_8134_Close(I32 cardNo) (All Windows)
U16 W_8134_Set_Config(char *fileName)
void W_8134_Get_IRQ_Channel(U16 cardNo, U16 *irq_no )
void W_8134_Get_Base_Addr(U16 cardNo, U16 *base_addr )
```

Visual Basic (Windows)

```
W_8134_Initial (existCards As Integer, pciInfo As
    PCI_INFO) As Integer (Windows 9x Only)
W_8134_Initial (ByVal cardNo As Long) As Integer
    (Windows NT/2K/XP)
W_8134_Close (ByVal cardNo As Long) As Integer
    (Windows NT/2K/XP)
W_8134_Set_Config (ByVal fileName As String) As
    Integer
W_8134_Get_IRQ_Channel    (ByVal cardno As Integer,
    irq_no As Integer)
W_8134_Get_Base_Addr     (ByVal cardno As Integer,
    base_addr As Integer)
```

@ *Argument*

existCards: numbers of existing PCI-8134 cards
info: relative information of the PCI-8134 cards
cardNo: The PCI-8134 card index number.
filename: A configuration file from MotionCreator
irq_no: The card's IRQ channel number
base_addr: The card's base address

@ *Return Code*

ERR_NoError
ERR_BoardNoInit
ERR_PCIBiosNotExist

6.4 Pulse Input / Output Configuration

@ Name

set_pls_outmode – Set the configuration for pulse command output.

set_pls_iptmode – Set the configuration for feedback pulse input.

set_cnt_src – Enable/Disable the external feedback pulse input

@ Description

set_pls_outmode:

Configure the output modes of command pulse. There are two modes for command pulse output.

set_pls_iptmode:

Configure the input modes of external feedback pulse. There are four types for feedback pulse input. Note that this function makes sense only when **cnt_src** parameter in **set_cnt_src()** function is enabled.

set_cnt_src:

If external encoder feedback is available in the system, set the **cnt_src** parameter in this function to *Enabled* state. Then internal 28-bit up/down counter will count according configuration of **set_pls_iptmode()** function. Or the counter will count the command pulse output.

@ Syntax

C/C++ (DOS, Windows)

```
U16 set_pls_outmode(I16 axis, I16 pls_outmode)
U16 set_pls_iptmode(I16 axis, I16 pls_iptmode)
U16 set_cnt_src(I16 axis, I16 cnt_src)
```

Visual Basic (Windows)

```
set_pls_outmode (ByVal axis As Long, ByVal pls_outmode
   As Long) As Integer
set_pls_iptmode (ByVal axis As Long, ByVal pls_iptmode
   As Long) As Integer
set_cnt_src (ByVal axis As Long, ByVal cnt_src As Long)
   As Integer
```

@ Argument

axis:axis number designated to configure pulse Input/Output.

pls_outmode: setting of command pulse output mode for OUT and DIR pins.
pls_outmode=0, OUT/DIR type pulse output.
pls_outmode=1, CW/CCW type pulse output.

pls_iptmode: setting of encoder feedback pulse input mode for EA and EB pins.
pls_iptmode=0, 1X AB phase type pulse input.
pls_iptmode=1, 2X AB phase type pulse input.
pls_iptmode=2, 4X AB phase type pulse input.

```
pls_iptmode=3, CW/CCW type pulse input.

cnt_src: Counter source
     cnt_src=0, counter source from command pulse
     cnt_src=1, counter source from external input
        EA, EB
```

@ Return Code

ERR_NoError

6.5 Continuously Motion Move

@ Name

v_move – Accelerate an axis to a constant velocity with trapezoidal profile

sv_move – Accelerate an axis to a constant velocity with S-curve profile

v_change – Change speed on the fly

v_stop – Decelerate to stop

set_sd_stop_mode – Set slow down stop mode

fix_max_speed – Fix maximum speed setting

unfix_max_speed – Unfix maximum speed setting

get_current_speed – Get axis' current output pulse rate

verify_speed – Get the minimum acceleration time under the speed setting

_8134_set_sd_stop_mode – Set slow down stop mode

@ Description

v_move:

This function is used to accelerate an axis to the specified constant velocity. The axis will continue to travel at a constant velocity until the velocity is changed or the axis is commanded to stop. The direction is determined by the sign of velocity parameter.

sv_move:

This function is similar to v_stop() but the accelerating is S-curve.

v_change:

You can change the velocity profile of command pulse ouput during operation by this function. This function changes the maximum velocity setting during operation.

v_stop:

This function is used to decelerate an axis to stop anytime.

set_sd_stop_mode:

Select the motion actions for slow down only or slow down then stop when SD pin is turned on

fix_max_speed:

This function is used to fix the speed resolution multiplier. The higher it is set, the coarser speed step is performed but the higher acceleration rate is obtained. Once it is set, the motion function will use this multiplier setting instead. Notice that this value will not affect the maximum speed of the motion command.

unfix_max_speed:

This function is used to unfix the speed resolution multiplier. Once it is unfixed, all motion command will calculate a optimized multiplier value according to the maximum speed setting in motion function.

verify_speed:

This function is used to get the minimum acceleration under a maximum speed setting. This function will not affect any speed or acceleration setting. It is only for offline checking.

_8134_set_sd_stop_mode:

Select the motion actions for slow down only or slow down then stop when SD pin is turned on

@ Syntax

C/C++ (DOS, Windows)

```
U16 v_move(I16 axis, F64 str_vel, F64 max_vel, F64
    Tacc)
U16 sv_move(I16 axis, F64 str_vel, F64 max_vel, F64
    Tlacc, F64
        Tsacc)
U16 v_change(I16 axis, F64 max_vel, F64 Tacc)
U16 v_stop(I16 axis, F64 Tdec)
```

U16 set_sd_stop_mode(I16 axis,I16 stop_mode)
U16 fix_max_speed(I16 axis, F64 max_vel)
U16 unfix_max_speed(I16 axis)
*F64 verify_speed(F64 StrVel, F64 MaxVel, F64 *minAccT, F64 *maxAccT, F64 MaxSpeed)*
I16 _8134_set_sd_stop_mode(I16 AxisNo, I16 sd_mode)

Visual Basic (Windows)

```
v_move (ByVal axis As Integer, ByVal str_vel As Double,
    ByVal max_vel As Double, ByVal Tacc As Double) As
    Integer
sv_move(I16 axis, F64 str_vel, F64 max_vel, F64 Tlacc,
    F64 Tsacc) As Integer
v_change(I16 axis, F64 max_vel, F64 Tacc) As Integer
```

v_stop (ByVal axis As Integer, ByVal Tacc As Double) As Integer *set_sd_stop_mode (ByVal axis As Integer, ByVal stopmode As Integer) As Integer*

fix_max_speed(ByVal axis As Integer, ByVal max_vel As Double) As Integer
unfix_max_speed(ByVal axis As Integer) As Integer
verify_speed(ByVal str_vel As Double, ByVal max_vel As Double, minAccT As Double, maxAccT As Double, ByVal MaxSpeed As Double) As Double
_8134_set_sd_stop_mode (ByVal axis As Integer, ByVal stopmode As Integer) As Integer

@ ***Argument***

axis: axis number designated to move or stop.
str_vel: starting velocity in unit of pulse per second
max_vel: maximum velocity in unit of pulse per second
Tacc: specified acceleration time in unit of second
Tdec: specified deceleration time in unit of second
Tlacc: specified linear acceleration time of S-curve in unit of second
Tsacc: specified S-curve acceleration time of S-curve in unit of second
stopmode: 0=slow down to starting velocity, 1=slow down then stop
***minAccT:** calculated minimum acceleration time
***maxAccT:** calculated maximum acceleration time
MaxSpeed: The value of maximum speed setting in motion function or in fix_max_speed function

@ ***Return Code***

ERR_NoError
The return value of verify_speed is the calculated starting velocity

---

6.6 Trapezoidal Motion Mode

@ ***Name***

start_a_move– Begin an absolute trapezoidal profile motion
start_r_move– Begin a relative trapezoidal profile motion
start_t_move– Begin a non-symmetrical relative trapezoidal profile motion
start_ta_move – Begin a non-symmetrical absolute trapezoidal profile motion

a_move– Begin an absolute trapezoidal profile motion and wait for completion

r_move– Begin a relative trapezoidal profile motion and wait for completion

t_move – Begin a non-symmetrical relative trapezoidal profile motion and wait for completion

ta_move– Begin a non-symmetrical absolute trapezoidal profile motion and wait for completion

wait_for_done – Wait for an axis to finish

set_rdp_mode – Set ramping down mode

@ Description

start_a_move() :

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the specified absolute position, immediately returning control to the program. The acceleration rate is equal to the deceleration rate. **a_move()** starts an absolute coordinate move and waits for completion.

start_r_move() :

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the relative distance, immediately returning control to the program. The acceleration rate is equal to the deceleration rate. **r_move()** starts a relative move and waits for completion.

start_ta_move() :

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the specified

absolute position, immediately returning control to the program.. **ta_move()** starts an absolute coordinate move and waits for completion.

start_t_move() :

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the relative distance, immediately returning control to the program.. **t_move()** starts a relative coordinate move and waits for completion.

The moving direction is determined by the sign of ***pos*** or ***dist*** parameter.If the moving distance is too short to reach the specified velocity, the controller will accelerate for the first half of the distance and decelerate for the second half (triangular profile). **wait_for_done()** waits for the motion to complete.

wait_for_done():

This function will return after the specified axis is not busy for motion.

set_rdp_mode():

Switch the motion slow down mode for auto or manual mode. The mode is default in manual mode.

@ Syntax

C/C++ (DOS, Windows)

```
U16 start_a_move(I16 axis, F64 pos, F64 str_vel, F64
    max_vel, F64 Tacc)
U16 a_move(I16 axis, F64 pos, F64 str_vel, F64 max_vel,
    F64Tacc)
U16 start_r_move(I16 axis, F64 distance, F64 str_vel,
    F64 max_vel, F64 Tacc)
U16 r_move(I16 axis, F64 distance, F64 str_vel, F64
    max_vel, F64Tacc)
U16 start_t_move(I16 axis, F64 dist, F64 str_vel, F64
    max_vel, F64 Tacc, F64 Tdec)
U16 t_move(I16 axis, F64 dist, F64 str_vel, F64
    max_vel, F64 Tacc, F64 Tdec)
U16 start_ta_move(I16 axis, F64 pos, F64 str_vel, F64
    max_vel, F64 Tacc, F64 Tdec)
U16 ta_move(I16 axis, F64 pos, F64 str_vel, F64
    max_vel, F64Tacc, F64 Tdec)
U16 wait_for_done(I16 axis)
U16 set_rdp_mode(I16 axisno, I16 mode)
```

Visual Basic (Windows)

```
start_a_move (ByVal axis As Integer, ByVal pos As
    Double, ByVal str_vel As Double, ByVal max_vel As
    Double, ByVal Tacc l As Double) As Integer
a_move (ByVal axis As Integer, ByVal pos As Double,
    ByVal str_vel As Double, ByVal max_vel As Double,
    ByVal Tacc As Double) As Integer
start_r_move (ByVal axis As Integer, ByVal distance As
    Double, ByVal str_vel As Double, ByVal max_vel As
    Double, ByVal Tacc As Double) As Integer
r_move (ByVal axis As Integer, ByVal distance As
    Double, ByVal str_vel As Double, ByVal max_vel As
    Double, ByVal Tacc As Double) As Integer
    start_t_move (ByVal axis As Integer, ByVal
        distance As Double, ByVal str_vel As Double,
        ByVal max_vel As Double, ByVal Tacc As
        Double, ByVal Tdec As Double) As Integer
t_move (ByVal axis As Integer, ByVal distance As
    Double, ByVal str_vel As Double, ByVal max_vel As
    Double, ByVal Tacc As Double, ByVal Tdec As Double)
    As Integer
start_ta_move(ByVal axis As Integer, ByVal pos As
    Double , ByVal str_vel As Double, ByVal max_vel As
    Double, ByVal Tacc As Double, ByVal Tdec As Double)
    As Interger ta_move(ByVal axis As Integer, ByVal
    pos As Double , ByVal str_vel As Double, ByVal
    max_vel As Double, ByVal Tacc As Double, ByVal Tdec
    As Double) As Integer
wait_for_done(ByVal axis As Integer) As Integer
set_rdp_mode(ByVal axisno As Integer, ByVal mode As
```

```
Integer) As Integer
```

@ ***Argument***

axis: axis number designated to move.
pos: specified absolute position to move
distance or dist: specified relative distance to move
str_vel: starting velocity of a velocity profile in unit of pulse per second
max_vel: starting velocity of a velocity profile in unit of pulse per second
Tacc: specified acceleration time in unit of second
Tdec: specified deceleration time in unit of second **Mode**: 0=Manual Mode(default), 1=Auto Mode

@ ***Return Code***

ERR_NoError
ERR_MoveError

6.7 S-Curve Profile Motion

@ ***Name***

start_s_move– Begin a S-Curve profile motion
s_move– Begin a S-Curve profile motion and wait for completion
start_rs_move– Begin a relative S-Curve profile motion
rs_move– Begin a relative S-Curve profile motion and wait for completion
start_tas_move– Begin a non-symmetrical absolute S-curve profile motion
tas_move– Begin a non-symmetrical absolute S-curve profile motion and wait for completion

@ ***Description***

start_s_move() :

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the specified absolute position, immediately returning control to the program. The acceleration rate is equal to the deceleration rate. **s_move()** starts an absolute coordinate move and waits for completion.

start_rs_move() :

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the relative distance, immediately returning control to the program. The acceleration rate is equal to the deceleration rate. **rs_move()** starts a relative move and waits for completion.

start_tas_move() :

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the specified absolute position, immediately returning control to the program. **tas_move()** starts an absolute coordinate move and waits for completion.

@ ***Syntax***

C/C++ (DOS, Windows)

```
U16 start_s_move(I16 axis, F64 pos, F64 str_vel, F64
    max_vel, F64 Tlacc, F64 Tsacc)
U16 s_move(I16 axis, F64 pos, F64 str_vel, F64 max_vel,
    F64 Tlacc, F64 Tsacc)
U16 start_rs_move(I16 axis, F64 distance, F64 str_vel,
    F64 max_vel, F64 Tlacc, F64 Tsacc)
U16 rs_move(I16 axis, F64 distance, F64 str_vel, F64
    max_vel, F64 Tlacc, F64 Tsacc)
U16 start_tas_move(I16 axis, F64 pos, F64 str_vel, F64
    max_vel, F64 Tlacc, F64 Tsacc, F64 Tldec, F64 Tsdec)
U16 tas_move(I16 axis, F64 pos, F64 str_vel, F64
    max_vel, F64 Tlacc, F64 Tsacc, F64 Tldec, F64 Tsdec)
```

Visual Basic (Windows)

```
start_s_move(ByVal axis As Integer, ByVal pos As
    Double, ByVal str_vel As Double, ByVal max_vel As
    Double, ByVal Tlacc As Double, ByVal Tsacc As
    Double) As Integer
s_move(ByVal axis As Integer, ByVal pos As Double,
    ByVal str_vel As Double, ByVal max_vel As Double
    ByVal Tlacc As Double, ByVal Tsacc As Double) As
    Integer
start_rs_move(ByVal axis As Integer, ByVal distance As
    Double, ByVal str_vel As Double, ByVal max_vel As
    Double, ByVal Tlacc As Double, ByVal Tsacc As
    Double) As Integer rs_move(ByVal axis As Integer,
    ByVal distance As Double, ByVal str_vel As Double,
    ByVal max_vel As Double, ByVal Tlacc As Double,
    ByVal Tsacc As Double) As Integer
    start_tas_move(ByVal axis As Integer, ByVal pos As
    Double, ByVal str_vel As Double, ByVal max_vel As
    Double, ByVal Tlacc As Double, ByVal Tsacc As
    Double, ByVal Tldec As Double, ByVal Tsdec As
    Double) As Integer
tas_move(ByVal axis As Integer, ByVal pos As Double
    ByVal str_vel As Double, ByVal max_vel As Double
    ByVal Tlacc As Double, ByVal Tsacc As Double, ByVal
    Tldec As Double, ByVal Tsdec As Double) As Integer
```

@ ***Argument***

axis: axis number designated to move.
pos: specified absolute position to move
distance or dist: specified relative distance to move
str_vel: starting velocity of a velocity profile in unit of pulse per second
max_vel: starting velocity of a velocity profile in unit of pulse per second
Tlacc: specified linear acceleration time in unit of second
Tsacc: specified S-curve acceleration time in unit of second
Tldec: specified linear deceleration time in unit of second

Tsdec: specified S-curve deceleration time in unit of second

@ ***Return CodeERR_NoError ERR_MoveError***

6.8 Multiple Axes Point to Point Motion

@ ***Name***

start_move_all– Begin a multi-axis trapezoidal profile motion

move_all– Begin a multi-axis trapezoidal profile motion and wait for completion

wait_for_all– Wait for all axes to finish

start_sa_move_all– Begin a multi-axis S-curve profile motion

move_all– Begin a multi-axis trapezoidal profile motion and wait for completion

@ ***Description***

start_move_all() :

This function causes the specified axes to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the specified absolute position, immediately returning control to the program. The move axes are specified by axes and the number of axes are defined by ***n_axes***. The acceleration rate of all axes is equal to the deceleration rate. **move_all()** starts the motion and waits for completion. Both functions guarantee that motion begins on all axes at the same sample time. **Note** that it is necessary to make connections according to Section 3.12 on CN4 if these two functions are needed.

wait_for_done() waits for the motion to complete for all of the specified axes. **start_sa_move_all()** is similar to this function.

The following code demos how to utilize these functions. This code moves axis 0 and axis 4 to position 8000.0 and 120000.0 respectively. If we choose velocities and acelerations that are propotional to the ratio of distances, then the axes will arrive at their endpoints at the same time (simultaneous motion).

```
#include   “pci_8134.h”

I16  axes [2] = {0, 4};
     F64
          positions[2] = {8000.0, 12000.0}; F64 str_vel[2]={0.0, 0.0};
          F64 max_vel[2]={4000.0, 6000.0}; F64 Tacc[2]]={0.04, 0.06};

move_all(2, axes, positions, str_vel, max_vel, Tacc);
```

@ ***Syntax***

C/C++ (DOS, Windows)

```
U16 start_move_all(I16 len, I16 *axes, F64 *pos, F64
    *str_vel, F64 *max_vel, F64 *Tacc)
```

```
U16 move_all(I16 len, I16 *axes, F64 *pos, F64
    *str_vel, F64 *max_vel, F64 *Tacc)
U16 wait_for_all(I16 len, I16 *axes)
U16 start_sa_move_all(I16 len, I16 *axes, F64 *pos,
F64 *str_vel, F64 *max_vel, F64 *Tlacc, F64 *Tsacc)
```

Visual Basic (Windows)

```
start_move_all(ByVal len As Integer, ByRef axis As
    Integer , ByRef pos As Double, ByRef str_vel As
    Double, ByRef max_vel As Double, ByRef Tacc As
    Double) As Integer
move_all(ByVal len As Integer, ByRef axis As Integer,
    ByRef pos As Double, ByRef str_vel As Double, ByRef
    max_vel As Double, ByRef Tacc As Double) As Integer
    wait_for_all(ByVal n_axes As Integer, ByRef axis As
    Integer) As Integer
start_move_all(ByVal len As Integer, ByRef axis As
Integer , ByRef pos As Double, ByRef str_vel As
Double, ByRef max_vel As Double, ByRef Tlacc As
Double, ByRef Tsacc As Double) As Integer
```

@ ***Argument***

n_axes: number of axes for simultaneous motion
***axes**: specified axes number array designated to move.
***pos**: specified position array in unit of pulse
***str_vel**: starting velocity array in unit of pulse per second
***max_vel**: maximum velocity array in unit of pulse per second
***Tacc**: acceleration time array in unit of second

@ ***Return Code***

ERR_NoError
ERR_MoveError

6.9 Linear Interpolated Motion

@ Name

move_xy – Perform a 2-axes linear interpolated motion between X & Y and wait for finish

move_zu – Perform a 2-axes linear interpolated motion between Z & U and wait for finish

start_move_xy – Start a 2-axes linear interpolated motion between X & Y

start_move_zu – Start a 2-axes linear interpolated motion between Z & U

@ Descriptionmove_xy, move_zu:

These two functions cause a linear interpolation motion between two axes and wait for completion. The moving speed should be set before performing these functions. Relations of speed between two axes are given in Chapter 4.1.4.

start_move_xy, start_move_zu:

These two functions cause a linear interpolation motion between two axes without waiting for completion. After issuing this function, it will start to move and leave the function at the same time.

@ Syntax

C/C++ (DOS, Windows)

```
U16 move_xy(I16 cardNo, F64 x, F64 y)
U16 move_zu(I16 cardNo, F64 z, F64 u)
U16 start_move_xy(I16 cardNo, F64 x, F64 y)
U16 start_move_zu(I16 cardNo, F64 z, F64 u)
```

Visual Basic (Windows)

```
move_xy (ByVal cardno As Long, ByVal x As Double,
   ByVal y As Double) As Integer
move_zu (ByVal cardno As Long, ByVal z As Double,
   ByVal u As Double) As Integer
start_move_xy (ByVal cardno As Long, ByVal x As Double,
   ByVal yAs Double) As Integer
start_move_zu (ByVal cardno As Long, ByVal z As
Double, ByVal u As Double) As Integer
```

@ Argument

cardNo: card number designated to perform interpolating function.

x, y, z, u: absolute target position of linear interpolation motion

@ Return Code

ERR_NoError

6.10 Interpolation Parameters Configuring

@ Name

map_axes – Configure the axis map for coordinated motion

set_move_speed – Set the vector velocity

set_move_accel – Set the vector acceleration time

set_move_ratio – Set the axis resolution ratios

_8134_set_move_ratio – Set the axis resolution ratios

@ Description

map_axes:

This function initializes a group of axes for coordinated motion. **map_axes()** must be called before any coordinated motion function is used. For PCI-8134, coordinated motion is made only between two axes. For example, if the z and u coordinates correspond to axes 2 and 3, the following code would be used to define the coordinate system:

```
int ax[2] = {2, 3};
map_axes(2, ax);
set_move_speed(10000.0);        // Set vector velocity = 10000pps
set_move_accel(0.1);   // Set vector accel. time = 0.1 sec
```

set_move_speed, set_move_accel:

The vector velocity and vector acceleration can be specified for coordinated motion by this two functions. Codes at last samples demonstrates how to utilize this two function associated with **map_axes()**.

set_move_ratio:

This function configures scale factors for the specified axis. Usually, the axes only need scale factors if their mechanical resolutions are different. For example, if the resolution of feedback sensors is two times resolution of command pulse, then ***ratio = 2***.

_8134_set_move_ratio:

This function configures scale factors for the specified axis. Usually, the axes only need scale factors if their mechanical resolutions are different. For example, if the resolution of feedback sensors is two times resolution of command pulse, then ***ratio = 2***.

@ Syntax

C/C++ (DOS, Windows)

```
U16 map_axes(U16 n_axes, U16 *map_array)
U16 set_move_speed(F64 str_vel, F64 max_vel)
U16 set_move_accel(F64 Tacc)
U16 set_move_ratio(U16 axis, F64 ratio)
I16 _8134_set_move_ratio(I16 AxisNo, F64 move_ratio)
```

Visual Basic (Windows)

```
map_axes (ByVal n_axes As Integer, map_array As
    Integer) As Integer
```

```
set_move_speed (ByVal str_vel As Double, ByVal max_vel
    As Double) As Integer
set_move_accel (ByVal accel As Double) As Integer
set_move_ratio (ByVal axis As Integer, ByVal ratio As
    Double) As Integer
_8134_set_move_ratio (ByVal axis As Integer, ByVal
    ratio As Double) As Integer
```

*@ **Argument***

axis: axis number designated to configure
n_axes: number of axes for coordinated motion
***map_array**: specified axes number array designated to move.
str_vel: starting velocity in unit of pulse per second **max_vel**: maximum velocity in unit of pulse per second **Tacc**: specified acceleration time in unit of second

ratio: ratio of (feedback resolution)/(command resolution)

*@ **Return Code***

ERR_NoError

6.11 Home Return

*@ **Name***

set_home_config – Set the configuration for home return.

home_move – Perform a home return move.

*@ **Description***

set_home_config:

Configure the logic of origin switch and index signal needed for home_move() function. If you need to stop the axis after EZ signal is active(home_mode=1 or 2), you should keep placing ORG signal in the ON status until the axis stop. If the pulse width of ORG signal is too short to keep it at ON status till EZ goes ON, you should select the org_latch as enable. The latched condition is cancelled by the next start or by disabling the org_latch. Three home return modes are available. Refer to Chapter4.1.5 for the setting of home_mode control.

home_move:

This function will cause the axis to perform a home return move according to the setting of **set_home_config()** function. The direction of moving is determined by the sign of velocity parameter(svel, mvel). Since the stopping condition of this function is determined by *home_mode* setting, user should take care to select the initial moving direction. Or user should take care to handle the condition when limit switch is touched or other conditions that is possible causing the axis to stop. Executing v_**stop()** function during **home_move()** can also cause the axis to stop.

*@ **Syntax***

C/C++ (DOS, Windows 95/NT)

```
U16 set_home_config(I16 axis, I16 home_mode, I16
    org_logic, I16 org_latch, I16 EZ_logic)
U16 home_move(I16 axis, F64 svel, F64 mvel, F64 accel)
```

Visual Basic (Windows)

```
set_home_config (ByVal axis As Long, ByVal home_mode
    As Long, ByVal org_logic As Long, ByVal org_latch
    As Long, ByVal EZ_logic As Long) As Integer
home_move (ByVal axis As Long, ByVal str_vel As Double,
    ByVal max_vel As Double, ByVal accel As Double) As
    Integer
```

@ ***Argument***

axis: axis number designated to configure and perform home returning

home_mode: stopping modes for home return.

home_mode=0, ORG active only.

home_mode=1, ORG active and then EZ active to stop.

home_mode=2, ORG active and then EZ active slow down to stop.

home_mode=3~7, please refer to the appendix A

org_logic: Action logic configuration for ORG signal

org_logic=0, active low; org_logic=1, active high

org_latch: Latch state control for ORG signal

org_latch=0, don't latch input; org_latch=1, latch input.

EZ_logic: Action logic configuration for EZ signal

EZ_logic=0, active low; EZ_logic=1, active high.

@ ***Return Code***

ERR_NoError

6.12 Manual Pulser Motion

@ ***Name***

set_manu_iptmode – Set pulser input mode and operation mode

manu_move – Begin a manual pulser movement

@ ***Description***

set_manu_iptmode:

Four types of pulse input modes can be available for pulser or hand wheel. User can also move two axes simultaneously with one pulser by selecting the operation mode to ***common mode***. Or move the axes independently by selecting the operation mode to ***independent mode***.

manu_move:

Begin to move the axis according to manual pulser input as this command is written. The maximum moving velocity is limited by **mvel** parameter. Not until the ***v_stop()*** command is written won't system end the manual move mode.

@ ***Syntax***

C/C++ (DOS, Windows)

```
U16 set_manu_iptmode(I16 axis, I16 ipt_mode, I16
    op_mode)
```

```
U16 manu_move(I16 axis, F64 mvel)
```

Visual Basic (Windows)

```
set_manu_iptmode (ByVal axis As Long, ByVal
    manu_iptmode As Long, ByVal op_mode As Long) As
    Integer
manu_move (ByVal axis As Long, ByVal max_vel As Double)
    As Integer
```

@ ***Argument***

axis: axis number designated to start manual move

ipt_mode: setting of manual pulser input mode from PA and PB pins
ipt_mode=0, 1X AB phase type pulse input.
ipt_mode=1, 2X AB phase type pulse input.
ipt_mode=2, 4X AB phase type pulse input.
ipt_mode=3, CW/CCW type pulse input.

op_mode: common or independent mode selection
op_mode=0, Independent for each axis
op_mode=1,PAX, PBX common for PAY, PBY
or PAZ, PBZ common for PAU, PBU.

mvel: limitation for maximum velocity

@ ***Return Code***

ERR_NoError

6.13 Motion Status

@ ***Name***

motion_done – Return the status when a motion is done

@ ***Description***

motion_done:

Return the motion status of PCI-8134.
Definition of return value is as following:

Return value =

0: the axis is busying.
1: a movement is finished
2: the axis stops at positive limit switch
3: the axis stops at negative limit switch
4: the axis stops at origin switch
5: the axis stops because the ALARM signal is active

The following code demonstrates how to utilize this function:

// Begin a trapezoidal velocity profile motion start_a_move(axis_x, pos1, svel, mvel, Tacc);

// Wait for completion while(!motion_done(axis_x*));*

@ ***Syntax***

C/C++ (DOS, Windows)

```
U16 motion_done(I16 axis)
```

Visual Basic (Windows)

```
motion_done (ByVal axis As Integer) As Integer
```

@ *Argument*

axis: axis number of motion status

@ *Return Code*

ERR_NoError

6.14 Servo Drive Interface

@ *Name*

set_alm_logic – Set alarm logic and alarm mode

set_inp_logic – Set In-Position logic and enable/disable

set_sd_logic – Set slow down point logic and enable/disable

set_erc_enable – Set ERC pin output enable/disable

@ Description set_alm_logic:

Set the active logic of **ALARM** signal input from servo driver. Two reacting modes are available when **ALARM** signal is active.

set_inp_logic:

Set the active logic of **In-Position** signal input from servo driver. Users can select whether they want to enable this function. Default state is disabled.

set_sd_logic:

Set the active logic and latch control of **SD** signal input from mechanical system. Users can select whether they want to enable this function. Default state is disabled.

set_erc_enable:

You can set ERC pin output enable/disable by this function. Default state is enabled.

@ *Syntax*

C/C++ (DOS, Windows)

```
U16   set_alm_logic(I16   axis,   I16   alm_logic,   I16
   alm_mode)
U16   set_inp_logic(I16   axis,   I16   inp_logic,   I16
   inp_enable)
U16 set_sd_logic(I16 axis, I16 sd_logic, I16 sd_latch,
   I16 sd_enable)
U16 set_erc_enable(I16 axis, I16 erc_enable)
```

Visual Basic (Windows)

```
set_alm_logic  (ByVal axis As Long, ByVal alm_logic As
   Long, ByVal alm_mode As Long) As Integer
set_inp_logic  (ByVal axis As Long, ByVal inp_logic As
   Long, ByVal inp_enable As Long) As Integer
set_sd_logic  (ByVal axis As Long, ByVal sd_logic As
   Long, , ByVal sd_latch As Long, ByVal sd_enable As
   Long) As Integer
```

```
set_erc_enable(ByVal axis As Integer, ByVal erc_enable
   As Long) As Integer
```

@ ***Argument***

axis: axis number designated to configure
alm_logic: setting of active logic for ALARM signal
alm_logic=0, active LOW.
alm_logic=1, active HIGH.
inp_logic: setting of active logic for INP signal
inp_logic=0, active LOW.
inp_logic=1, active HIGH.
sd_logic: setting of active logic for SD signal
sd_logic=0, active LOW.
sd_logic=1, active HIGH.
sd_latch: setting of latch control for SD signal
sd_logic=0, do not latch.
sd_logic=1, latch.

alm_mode: reacting modes when receiving ALARM signal.
alm_mode=0, motor immediately stops.
alm_mode=1, motor decelerates then stops.
inp_enable: INP function enable/disable
inp_enable=0, Disabled
inp_enable=1, Enabled
sd_enable: Slow down point function enable/disable
sd_enable=0, Disabled
sd_enable=1, Enabled
erc_enable: ERC pin output enable/disable
erc_enable=0, Disabled
erc_enable=1, Enabled

@ ***Return Code***

ERR_NoError

6.15 I/O Control and Monitoring

@ ***Name***

W_8134_Set_SVON – Set state of general purpose output pin
get_io_status – Get all the I/O status of PCI-8134

@ ***Description***

W_8134_Set_SVON:

Set the High/Low output state of general purpose output pin **SVON**.

get_io_status:

Get all the I/O status for each axis. The definition for each bit is as following:

Bit	Name	Description
0	+EL	Positive Limit Switch
1	-EL	Negative Limit Switch
2	+SD	Positive Slow Down Point
3	-SD	Negative Slow Down Point
4	ORG	Origin Switch
5	EZ	Index signal
6	ALM	Alarm Signal
7	SVON	SVON of PCL5023 pin output
8	RDY	RDY pin input
9	INT	Interrupt status
10	ERC	ERC pin output
11	INP	In-Position signal input

@ ***Syntax***

C/C++ (DOS)

```
U16 _8134_Set_SVON(I16 axis, I16 on_off)
U16 get_io_status(I16 axis, U16 *io_status)
```

C/C++ (Windows)

```
U16 W_8134_Set_SVON(I16 axis, I16 on_off)
U16 get_io_status(I16 axis, U16 *io_status)
```

Visual Basic (Windows)

```
W_8134_Set_SVON  (ByVal axis As Long, ByVal on_off As
   Long) As Integer
get_io_status   (ByVal  axis  As  Integer,  io_sts  As
   Integer) As Integer
```

@ ***Argument***

axis: axis number for I/O control and monitoring
on_off: setting for SVON pin digital output
on_off=0, SVON is LOW.
on_off=1, SVON is HIGH.
***io_status**: I/O status word. Where “1’ is ON and “0” is OFF. ON/OFF state is read based on the corresponding set logic.

@ ***Return Code***

ERR_NoError

6.16 Position Control

@ ***Name***

set_position – Set the actual position.
get_position – Get the actual position.

set_command – Set the current command position.

get_command – Get the current command position.

_8134_get_target_pos – Get the current command position.

_8134_reset_target_pos – Set the current command position.

@ Descriptionset_position()

changes the current actual position to the specified position.

get_position()

reads the current actual position. Note that when feedback signals is not available in the system, thus external encoder feedback is *Disabled* in **set_cnt_src()** function, the value gotten from this function is command position.

set_command()

changes the command position to the specified command position. The command position is the target position of this command, not the current command position.

get_command()

reads the current command position.

The command position is the target position of this command, not the current command position.

_8134_get_target_pos()

reads the current command position.

The command position is the target position of this command, not the current command position.

_8134_reset_target_pos()

changes the command position to the specified command position. The command position is the target position of this command, not the current command position.

@ Syntax

C/C++ (DOS, Windows)

```
U16 set_position(I16 axis, F64 pos)
U16 get_position(I16 axis, F64 *pos)
U16 set_command(I16 axis, F64 pos)
U16 get_command(I16 axis, F64 *pos)
I16 _8134_get_target_pos(I16 AxisNo, F64
*pos)
I16  _8134_reset_target_pos(I16  AxisNo,
F64 pos)
```

Visual Basic (Windows)

```
get_position  (ByVal axis As Integer, pos As Double)
   As Integer
set_position   (ByVal axis As Integer, ByVal pos As
   Double) As
      Integer
get_command  (ByVal axis As Integer, pos As Double) As
   Integer
set_command   (ByVal axis As Integer, ByVal pos As
   Double) As
```

Integer

```
_8134_get_target_pos  (ByVal axis As Integer, pos As
Double) As Integer
_8134_reset_target_pos  (ByVal axis As Integer, ByVal
pos As Double) As Integer
```

@ ***Argument***

axis: axis number designated to set and get position.

pos: actual position or command position

@ ***Return Code***

ERR_NoError

6.17 Interrupt Control

@ ***Name***

W_8134_INT_ENABLE – Set interrupt enable

W_8134_INT_Enable – Set interrupt enable

W_8134_INT_Disable – Set interrupt disable

W_8134_Set_INT_Control – Set interrupt event handle

set_int_factor – Set interrupt generating factorsget_int_status – Get the interrupting status of axis link_axis_interrupt – Create a interrupt ***callback*** function

@ ***Description***

W_8134_INT_Enable:

This function is used to enable interrupt event generating to host PC. **(**Window only**).**

W_8134_INT_Disable:

This function is used to disable interrupt event generating to host PC. **(**Window only**).**

W_8134_Set_INT_Control :

This function is used to control the hardware interrupt channel enable or disable. Please call this function after the interrupt events are enabled.

set_int_factor:

This function allows users to select factors to initiate the INT signal. PCI-8134 can generate INT signal to host PC by setting the relative bit as 1. The definition for each bit is as following:

Bit	Interrupt Factor
0	Stop with the EL signal
1	Stop with the SD signal
2	Stop with the ALM signal
3	Stop with the STP signal
4	Should be set to 0
5	Completion of home return

6	Completion of preset movement
7	Completion of interpolating motion for two axes: (X & Y) or (Z & U)
8~12	X (should be set to 0)
13	when ***v_stop()*** function stop the axis
14	EA/EB, PA/PB encoder input error
15	start with STA signal
16	Completion of acceleration
17	Start of deceleration
18~22	Should be Set to 0
23	RDY active (AP3 of PCL5023 change from 1 to 0)
24~31	Should be set to 0

Note: Bit 14: The interrupt is generated when pins EA and EB, or PA and PB change simultaneously. It means there has an encoder input error.

get_int_axis:

This function allows user to identify which axis generates the INT signal to host PC. (**DOS only**)

get_int_status:

This function allows user to identify what kinds of interrupt is generated. After user gets this value, the status register will be cleared to 0. The return value is a 32 bits unsigned integer and the definition for each bit is as following:

Bit	Interrupt Type
0	Stop with the +EL signal
1	Stop with the –EL signal
2	Stop with the +SD signal
3	Stop with the –SD signal
4	Stop with the ALM signal
5	Stop with the STP signal
6	Always 0
7	Always 0
8	Stop with v_stop() command
9	Stop with home return completed
10	Always 0
11	Stop with preset movement completed
12	Stop with EA/EB input error
13	Always 0
14	Stop with PA/PB input error

15	Start with STA signal
16	Acceleration Completed
17	Deceleration Started
18~22	Always 0
23	RDY active (AP3 of PCL5023 change from 1 to 0)
24~31	Always 0

link_axis_interrupt:

This function is used to create a callback function in Windows for interrupt signal receiving. Once the interrupt comes, the callback function will be called too.

@ Syntax

C/C++ (DOS)

```
U16 _8134_Set_INT_ENABLE(U16 cardNo, U16 intFlag)
U16 set_int_factor(U16 axis, U32 int_factor)
U16 get_int_axis(U16 *int_axis)
U16 get_int_status(U16 axis, U32 *int_status)
```

C/C++ (Windows)

```
U16 W_8134_INT_Enable (I16 cardNo, HANDLE *phEvent)
U16 W_8134_INT_Disable (I16 cardNo)
void W_8134_Set_INT_Control(U16 cardNo, U16 intFlag)
U16 set_int_factor(U16 axis, U32 int_factor)
U16 get_int_status(I16 axis, U32 *int_status)
```

*I16 link_axis_interrupt(I16 AxisNo, void (_stdcall *callbackAddr)(void))*

Visual Basic (Windows)

```
W_8134_INT_Enable  (ByVal cardNo As Long, phEvent As Long)
W_8134_INT_Disable  (ByVal cardNo As Long) As Integer
W_8134_Set_INT_Control    (ByVal cardno As Integer, ByVal intFlag As Integer)
set_int_factor     (ByVal axis As Integer, ByVal int_factor As Long) As Integer
get_int_status   (ByVal axis As Long, int_status As Long) As Integer
```

link_axis_interrupt(ByVal AxisNo As Integer, By Val callbackAddr as Long) As Integer

@ ***Argument***

cardNo: card number 0,1,2,3…
axis: axis number 0,1,2,3,4…
intFlag: int flag, 0 or 1
phEvent: event or event array for interrupt axis (For Windows only)
int_factor: interrupt factor, refer to previous interrupt factor table
int_axis: interrupt axis number (the return value)
int_status: interrupt factor (the return value), refer to previous interrupt type table
callbackAddr: The call back function address

@ ***Return Code***

ERR_NoError

7

Additional Function Library (8134A.DLL)

This chapter describes the another supporting software for PCI-8134 cards. It is called 8134A.LIB and 8134A.DLL. Notice that this function library can't not be mixed to use with chapter 6, 8134.DLL. Users can use these functions to develop their application programs in C or Visual Basic or C++ language.

7.1 List of Functions

Initialization	**Section 7.3**
_8134_initial	Card initialization
_8134_close	Card Close
_8134_config_from_file	Configure card according to Motion Creator's setting
_8134_get_irq_channel	Get IRQ channel
_8134_get_base_addr	Get Base Address
_8134_version_info	Get card's hardware and software/driver version

Pulse Input/Output Configuration	**Section 7.4**
_8134_set_pls_outmode	Set pulse command output mode
_8134_set_pls_iptmode	Set encoder input mode
_8134_set_feedback_src	Set feedback counter input source

Continuously Motion Mode	**Section 7.5**
_8134_tv_move	Accelerate an axis to a constant velocity with trapezoidal profile

_8134_sv_move	Accelerate an axis to a constant velocity with S-curve profile
_8134_v_change	Change speed on the fly
_8134_sd_stop	Decelerate to stop
_8134_emg_stop	Emergency Stop
_8134_set_sd	Set slow down stop mode and SD logic
_8134_fix_speed_range	Fix speed range setting
_8134_unfix_speed_range	Unfix speed range setting
_8134_get_current_speed	Get current speed in pps
_8134_verify_speed	Get the minimum acceleration time under the speed setting

Trapezoidal Motion Mode **Section 7.6**

_8134_start_ta_move	Start an absolute trapezidal profile move
_8134_start_tr_move	Start a relative trapezoidal profile move
_8134_set_rdp_mode	Set Ramping down mode

S-Curve Profile Motion **Section 7.7**

_8134_start_sa_move	Start an absolute S-curve profile move
_8134_start_sr_move	Start a relative S-curve profile move

Multiple Axes Point to Point Motion **Section 7.8**

_8134_start_move_all	Start a multi-axis profile move
_8134_stop_move_all	Stop a multi-axis profile move
_8134_set_tr_move_all	Set a multi-axis relative trapezoidal profile move
_8134_set_ta_move_all	Set a multi-axis absolute trapezoidal profile move
_8134_set_sr_move_all	Set a multi-axis relative S-curve profile move
_8134_set_sa_move_all	Set a multi-axis absolute trapezoidal profile move

Linear / Circular Interpolated Motion **Section 7.9**

_8134_start_tr_move_xy	Start a 2-axis linear interpolated move for X & Y
_8134_start_ta_move_xy	Start a 2-axis linear interpolated move for X & Y
_8134_start_sr_move_xy	Start a 2-axis linear interpolated move for X & Y
_8134_start_sa_move_xy	Start a 2-axis linear interpolated move for X & Y
_8134_start_tr_move_zu	Start a 2-axis linear interpolated move for Z & U
_8134_start_ta_move_zu	Start a 2-axis linear interpolated move for Z & U
_8134_start_sr_move_zu	Start a 2-axis linear interpolated move for Z & U
_8134_start_sa_move_zu	Start a 2-axis linear interpolated move for Z & U

Home Return Mode **Section 7.10**

_8134_set_home_config	Set or get the home/index logic configuration
_8134_home_move	Start a home return action
_8134_set_org_offset	Set ORG length
_8134_set_org_logic	Set ORG logic
_8134_set_bounce_filter	Set a bounce filter when homing

Manual Pulser Motion **Section 7.11**

_8134_set_pulser_iptmode	Set pulser input mode and operation mode
_8134_pulser_v_move	Begin a manual pulser movement

Motion Status **Section 7.12**

_8134_motion_done	Check if the axis is in motion

Servo Driver Interface **Section 7.13**

_8134_set_alm	Set alarm logic and alarm mode
8134_set_inp	Set In-Position logic and enable/disable
_8134_set_erc_enable	Set the ERC output enable/disable

I/O Control and Monitoring **Section 7.14**

_8134_set_servo	Set the output pin for servo ON conotrl
_8134_get_io_status	Get I/O staus

Position Counter Control **Section 7.15**

_8134_set_position	Set current position counter value
_8134_get_position	Get current position counter value
_8134_set_command	Set current command target value
_8134_get_command	Get current command target value
_8134_get_error_counter	Get current error counter value
_8134_reset_error_counter	Reset error counter value
_8134_set_feedback_error_detect	Set feedback error detect value

Interrupt Control **Section 7.16**

8134_int_enable	Set Interrupt Event enable
_8134_int_disable	Set Interrupt Event enable
_8134_int_control	Enable/Disable IRQ channel
_8134_set_int_factor	Set Interrupt generationg factors
_8134_get_int_status	Get the interrupting status of axis
_8134_link_axis_interrupt	Link a callback function for interrupt

7.2 C/C++ Programming Library

This section gives the details of all the functions. The function prototypes and some common data types are decelerated in **PCI-8134.H**. These data types are used by PCI-8134 library. We suggest you to use these data types in your application programs. The following table shows the data type names and their range.

Type Name	Description	Range
U8	8-bit ASCII character	0 to 255
I16	16-bit signed integer	-32768 to 32767

U16	16-bit unsigned integer	0 to 65535
I32	32-bit signed long integer	-2147483648 to 2147483647
U32	32-bit unsigned long integer	0 to 4294967295
F32	32-bit single-precision floating-point	-3.402823E38 to 3.402823E38
F64	64-bit double-precision floating-point	-1.797683134862315E308 to 1.797683134862315E309
Boolean	Boolean logic value	TRUE, FALSE

7.3 Initialization

@ Name

_8134_initial – Card Initialization
_8134_close – Card Close
_8134_config_from_file – Configure Card according to Motion Creator's save file
_8134_get_irq_channel – Get the card's IRQ number
_8134_get_base_addr – Get the card's base address
_8134_version_info – Get the card's version information

@ Description

_8134_initial:

This function is used to initialize PCI-8134 card. It has to be initialized by this function before calling other functions.

_8134_close:

This function must be called before the program ends.

_8134_config_from_file:

This function is used to configure PCI-8134 card. All the I/O configurations and some operating modes appeared on "Axis Configuration Window" of Motion Creator will be set to PCI-8134. Click "Save Configuration" button on the "Axis Configuration Window" if you want to use this function in the application program. Click "Save Configuration" button will save all the configurations to a file call "*8134.cfg*". This file will appear in the Windows' system directory.

_8134_get_irq_channel:

This function is used to get the PCI-8134 card's IRQ number.

_8134_get_base_addr:

This function is used to get the PCI-8134 card's base address.

_8134_version_info:

This function is used to get the PCI-8134 card's version information including hardware, software and device driver.

@ Syntax

C/C++ (DOS,Windows)

*I16 _8134_initial(I16 *existCards)*
I16 _8134_close(void)
*I16 _8134_config_from_file(U8 *fileName)*
*I16 _8134_get_irq_channel(I16 cardNo, U16 *irq_no)*
*I16 _8134_get_base_addr(I16 cardNo, U16 *base_addr)*
*I16 _8134_version_info(I16 CardNo, U16 *HardwareInfo, I32 *SoftwareInfo, I32 *DriverInfo)*

Visual Basic (Windows)

B_8134_initial (existCards As Integer) As Integer
B_8134_close () As Integer
B_8134_config_from_file (ByVal fileName As String) As Integer
B_8134_get_irq_channel (ByVal cardno As Integer, irq_no As Integer) As Integer
B_8134_get_base_addr (ByVal cardno As Integer, base_addr As Integer) As Integer
B_8134_version_info (ByVal CardNo As Integer, HardwareInfo As Integer, SoftwareInfo As Long, DriverInfo As Long)

@ ***Argument***

existCards: numbers of existing PCI-8134 cards
cardNo: The PCI-8134 card index number.
filename: A configuration file from MotionCreator
irq_no: The card's IRQ channel number
base_addr: The card's base address
HardwareInfo: 0x1000 in heximal
SoftwareInfo: Format=OS/YY/MM/DD in decimal
OS=00, Win32
OS=12, WinCE
OS=24, DOS
OS=36, DOSExt
OS=48, Linux
DriverInfor: The same with SoftwareInfo

@ ***Return Code*** ERR_NoError
ERR_BoardNoInit
ERR_PCIBiosNotExist

7.4 Pulse Input / Output Configuration

@ ***Name***

_8134_set_pls_outmode – Set the configuration for pulse command output.
_8134_set_pls_iptmode – Set the configuration for feedback pulse input.
_8134_set_feedback_src – Select feedback counter source

@ ***Description***

_8134_set_pls_outmode:

Configure the output modes of command pulse. There are two modes for command pulse output.

_8134_set_pls_iptmode:

Configure the input modes of external feedback pulse. There are four types for feedback pulse input. Note that this function makes sense only when using external feedback counter source.

_8134_set_feedback_src:

If external encoder feedback is available in the system, set the **src** parameter in this function to *Enabled* state. Then internal 28-bit up/down counter will count according configuration of _8134**_set_pls_iptmode()** function. Or the counter will count the command pulse output.

@ ***Syntax***

C/C++ (DOS, Windows)

I16 _8134_set_pls_outmode(I16 axis, I16 pls_outmode)
I16 _8134_set_pls_iptmode(I16 axis, I16 pls_iptmode)
I16 _8134_set_cnt_src(I16 axis, I16 src)

Visual Basic (Windows)

B_8134_set_pls_outmode (ByVal axis As Long, ByVal pls_outmode As Long) As Integer
B_8134_set_pls_iptmode (ByVal axis As Long, ByVal pls_iptmode As Long) As Integer
B_8134_set_feedback_src (ByVal axis As Long, ByVal src As Long) As Integer

@ ***Argument***

axis:axis number designated to configure pulse Input/Output.

pls_outmode: setting of command pulse output mode for OUT and DIR pins.
pls_outmode=0, OUT/DIR type pulse output.
pls_outmode=1, CW/CCW type pulse output.

pls_iptmode: setting of encoder feedback pulse input mode for EA and EB pins.
pls_iptmode=0, 1X AB phase type pulse input.
pls_iptmode=1, 2X AB phase type pulse input.
pls_iptmode=2, 4X AB phase type pulse input.
pls_iptmode=3, CW/CCW type pulse input.

src: Feedback Counter source
cnt_src=0, counter source from command pulse
cnt_src=1, counter source from external input EA, EB

@ ***Return Code***

ERR_NoError

7.5 Continuously Motion Move

@ Name

_8134_tv_move – Accelerate an axis to a constant velocity with trapezoidal profile

_8134_sv_move – Accelerate an axis to a constant velocity with S-curve profile

_8134_v_change – Change speed on the fly

_8134_sd_stop – Decelerate to stop

_8134_emg_stop – Immediately Stop

_8134_set_sd – Set slow down stop mode and logic

_8134_fix_speed_range – Fix speed range setting

_8134_unfix_speed_range – Unfix speed range setting

_8134_get_current_speed – Get axis' current output pulse rate

_8134_verify_speed – Get the minimum acceleration time under the speed setting

@ Description

_8134_tv_move:

This function is used to accelerate an axis to the specified constant velocity. The axis will continue to travel at a constant velocity until the velocity is changed or the axis is commanded to stop. The direction is determined by the sign of velocity parameter.

_8134_sv_move:

This function is similar to v_stop() but the accelerating is S-curve.

_8134_v_change:

You can change the velocity profile of command pulse ouput during operation by this function. This function changes the maximum velocity setting during operation.

_8134_sd_stop:

This function is used to decelerate an axis to stop anytime.

_8134_set_sd:

Select the motion actions for slow down only or slow down then stop when SD pin is turned on

_8134_fix_speed_range:

This function is used to fix the speed resolution multiplier. The higher it is set, the coarser speed step is performed but the higher acceleration rate is obtained. Once it is set, the motion function will use this multiplier setting instead. Notice that this value will not affect the maximum speed of the motion command.

_8134_unfix_speed_range:

This function is used to unfix the speed resolution multiplier. Once it is unfixed, all motion command will calculate a optimized multiplier

value according to the maximum speed setting in motion function.

_8134_verify_speed:

This function is used to get the minimum acceleration under a maximum speed setting. This function will not affect any speed or acceleration setting. It is only for offline checking.

_8134_get_current_speed:

This function is used to get the current speed value in pps of position counter

@ Syntax

C/C++ (DOS, Windows)

I16 _8134_tv_move(I16 axis, F64 str_vel, F64 max_vel, F64 Tacc)
I16 _8134_sv_move(I16 axis, F64 str_vel, F64 max_vel, F64 Tlacc, F64 Tsacc)
I16 _8134_v_change(I16 axis, F64 max_vel, F64 Tacc)
I16 _8134_sd_stop(I16 axis, F64 Tdec)
I16 _8134_emg_stop(I16 axis)
I16 _8134_set_sd(I16 axis,I16 enable, I16 sd_logic, I16 sd_latch, I16 stop_mode)
I16 _8134_fix_speed_range(I16 axis, F64 max_vel)
I16 _8134_unfix_speed_range(I16 axis)
*F64 _8134_verify_speed(F64 StrVel, F64 MaxVel, F64 *minAccT, F64 *maxAccT, F64 MaxSpeed)*

Visual Basic (Windows)

B_8134_tv_move (ByVal axis As Integer, ByVal str_vel As Double, ByVal max_vel As Double, ByVal Tacc As Double) As Integer
B_8134_sv_move(I16 axis, F64 str_vel, F64 max_vel, F64 Tlacc, F64 Tsacc) As Integer
B_8134_v_change(I16 axis, F64 max_vel, F64 Tacc) As Integer
B_8134_sd_stop (ByVal axis As Integer, ByVal Tacc As Double) As Integer
B_8134_emg_stop (ByVal axis As Integer) As Integer
B_8134_set_sd (ByVal axis As Integer, ByVal enable As Integer ByVal sd_logic As Integer ByVal sd_latch As Integer, ByVal stop_mode As Integer) As Integer
B_8134_fix_speed_range(ByVal axis As Integer, ByVal max_vel As Double) As Integer
B_8134_unfix_speed_range(ByVal axis As Integer) As Integer
B_8134_verify_speed(ByVal str_vel As Double, ByVal max_vel As Double, minAccT As Double, maxAccT As Double, ByVal MaxSpeed As Double) As Double

@ Argument

axis: axis number designated to move or stop.
str_vel: starting velocity in unit of pulse per second
max_vel: maximum velocity in unit of pulse per

second
Tacc: specified acceleration time in unit of second
Tdec: specified deceleration time in unit of second

Tlacc: specified linear acceleration time of S-curve in unit of second
Tsacc: specified S-curve acceleration time of S-curve in unit of second
stopmode: 0=slow down to starting velocity, 1=slow down then stop
***minAccT:** calculated minimum acceleration time
***maxAccT:** calculated maximum acceleration time
MaxSpeed: The value of maximum speed setting in motion function or in fix_max_speed function
enable: Enable or disable SD function
sd_logic: SD input logic setting
sd_latch: SD latch function on=1/off=0

@ Return Code

ERR_NoError
The return value of verify_speed is the calculated starting velocity

---

7.6 Trapezoidal Motion Mode

@ Name

_8134_start_ta_move – Begin an absolute trapezoidal profile motion
_8134_start_tr_move – Begin a relative trapezoidal profile motion
_8134_set_rdp_mode – Set ramping down mode

@ Description

_8134_start_ta_move() :

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the specified absolute position, immediately returning control to the program. The acceleration rate is equal to the deceleration rate.

_8134_start_tr_move() :

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the relative distance, immediately returning control to the program. The acceleration rate is equal to the deceleration rate.

_8134_set_rdp_mode():

Switch the motion slow down mode for auto or manual mode. The mode is default in manual mode.

@ Syntax

C/C++ (DOS, Windows)

I16 _8134_start_ta_move(I16 axis, F64 pos, F64 str_vel, F64 max_vel, F64 Tacc,F64 Tdec)

I16 _8134_start_tr_move(I16 axis, F64 distance, F64 str_vel, F64 max_vel, F64 Tacc,F64 Tdec)

I16 _8134_set_rdp_mode(I16 axisno, I16 mode)

Visual Basic (Windows)

B_8134_start_ta_move (ByVal axis As Integer, ByVal pos As Double, ByVal str_vel As Double, ByVal max_vel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8134_start_tr_move (ByVal axis As Integer, ByVal distance As Double, ByVal str_vel As Double, ByVal max_vel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8134_set_rdp_mode(ByVal axisno As Integer, ByVal mode As Integer) As Integer

@ ***Argument***

axis: axis number designated to move. **pos**:
specified absolute position to move **distance**:
specified relative distance to move
str_vel: starting velocity of a velocity profile in unit of pulse per second
max_vel: starting velocity of a velocity profile in unit of pulse per second
Tacc: specified acceleration time in unit of second
Tdec: specified deceleration time in unit of second
Mode: 0=Manual Mode(default), 1=Auto Mode

@ ***Return Code***

ERR_NoError
ERR_MoveError

7.7 S-Curve Profile Motion

@ ***Name***

_8134_start_sa_move– Start an absolute S-curve profile motion
_8134_start_sr_move– Start a relative S-curve profile motion

@ ***Description***

_8134_start_sa_move() :

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the specified absolute position, immediately returning control to the program.

_8134_start_sr_move() :

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the specified relative distance, immediately returning control to the program.

*@ **Syntax***

C/C++ (DOS, Windows)

I16 _8134_start_sa_move(I16 axis, F64 pos, F64 str_vel, F64 max_vel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec)

I16 _8134_start_sr_move(I16 axis, F64 distance, F64 str_vel, F64 max_vel, F64 Tacc, F64 Tdec, F64 SVdec, F64 SVacc)

Visual Basic (Windows)

B_8134_start_sr_move(ByVal axis As Integer, ByVal pos As Double, ByVal str_vel As Double, ByVal max_vel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal SVacc As Double, ByVal SVdec As Double) As Integer

B_8134_start_sa_move(ByVal axis As Integer, ByVal pos As Double ByVal str_vel As Double, ByVal max_vel As Double ByVal Tacc As Double, ByVal Tdec As Double, ByVal SVacc As Double, ByVal SVdec As Double) As Integer

*@ **Argument***

axis: axis number designated to move. **pos**:
specified absolute position to move **distance**:
specified relative distance to move
str_vel: starting velocity of a velocity profile in unit of pulse per second
max_vel: starting velocity of a velocity profile in unit of pulse per second
Tacc: specified total acceleration time in unit of second
Tdec: specified total deceleration time in unit of second
SVacc: specified S-curve acceleration range in unit of pps, default is 0
SVdec: specified S-curve deceleration range in unit of pps, default is 0

*@ **Return Code***

ERR_NoError
ERR_MoveError

7.8 Multiple Axes Point to Point Motion

*@ **Name***

_8134_set_tr_move_all – Multi-axis simultaneous operation setup.
_8134_set_ta_move_all – Multi-axis simultaneous operation setup.
_8134_set_sr_move_all – Multi-axis simultaneous operation setup.
_8134_set_sa_move_all – Multi-axis simultaneous operation setup.
_8134_start_move_all – Begin a multi-axis trapezoidal profile motion
_8134_stop_move_all –Simultaneously stop Multi-axis motion

*@ **Description***

Theses functions are related to simultaneous operations of multi-

axes, even in different cards. The simultaneous multi-axis operation means to start or stop moving specified axes at the same time. The axes moved are specified by the parameter **"AxisArray,"** and the number of axes are defined by parameter "***TotalAxes"*** in **_8134_set_tr_move_all()**.
When properly setup with **_8134_set_xx_move_all()**, the function **_8134_start_move_all()** will cause all specified axes to begin a trapezoidal relative movement, and _8134_stop_move_all() will stop them. Both functions guarantee that motion Start/Stop on all specified axes are at the same time. Note that it is necessary to make connections according to Section 3.12 on CN4 if the start/stop axes are on different cards.

The following code demos how to utilize these functions. This code moves axis 0 and axis 4 to distance 8000.0 and 120000.0 respectively. If we choose velocities and accelerations that are proportional to the ratio of distances, then the axes will arrive at their endpoints at the same time.

```
I16 axes[2] = {0, 4};
F64 dist[2] = {8000.0, 12000.0};
F64 str_vel[2]={0.0, 0.0};
F64 max_vel[2]={4000.0, 6000.0};
F64 Tacc[2]={0.04, 0.06};
F64 Tdec[2]= {0.04, 0.06};
_8134_set_tr_move_all(2, axes, dist, str_vel, max_vel, Tacc, Tdec);
_8134_start_move_all(axes[0]);
```

@ Syntax

C/C++ (DOS, Windows)

```
I16 _8134_set_tr_move_all(I16 TotalAxes, I16 *AxisArray, F64
    *DistA, F64 *StrVelA, F64 *MaxVelA, F64 *TaccA, F64 *TdecA)
I16 _8134_set_sa_move_all(I16 TotalAx, I16 *AxisArray, F64
    *PosA, F64 *StrVelA, F64 *MaxVelA, F64 *TaccA, F64 *TdecA,
    F64 *SVaccA, F64 *SVdecA)
I16 _8134_set_ta_move_all(I16 TotalAx, I16 *AxisArray, F64
    *PosA, F64 *StrVelA, F64 *MaxVelA, F64 *TaccA, F64 *TdecA)
I16 _8134_set_sr_move_all(I16 TotalAx, I16 *AxisArray, F64
    *DistA, F64 *StrVelA, F64 *MaxVelA, F64 *TaccA, F64 *TdecA,
    F64 *SVaccA, F64 *SvdecA)
I16 _8134_start_move_all(I16 FirstAxisNo)
I16 _8134_stop_move_all(I16 FirstAxisNo)
```

Visual Basic (Windows)

```
B_8134_set_tr_move_all(ByVal TotalAxes As Integer, AxisArray As
    Integer, DistA As Double, StrVelA As double, MaxVelA As
    double, TaccA As double, TdecA As double)
```

B_8134_set_sa_move_all(ByVal TotalAxes As Integer, AxisArray As Integer, PosA As Double, StrVelA As double, MaxVelA As double, TaccA As double, TdecA As double, SVaccA As double, SVdecA As Double)

B_8134_set_ta_move_all(ByVal TotalAxes As Integer, AxisArray As Integer, PosA As Double, StrVelA As double, MaxVelA As double, TaccA As double, TdecA As double)

B_8134_set_sr_move_all(ByVal TotalAxes As Integer, AxisArray As Integer, DistA As Double, StrVelA As double, MaxVelA As double, TaccA As double, TdecA As double, SVaccA As double, SVdecA As Double)

B_8134_start_move_all(ByVal FirstAxisNo As Integer)

B_8134_stop_move_all(ByVal FirstAxisNo As Integer)

@ ***Argument***

TotalAxes: number of axes for simultaneous motion
*****AxisArray: specified axes number array designated to move.
*****PosA: specified position array in unit of pulse
*****StrVelA: starting velocity array in unit of pulse per second
*****MaxVelA: maximum velocity array in unit of pulse per second
*****TaccA: acceleration time array in unit of second
*****TdecA: acceleration time array in unit of second
*SVaccA: acceleration array of S-curve part in unit of pps
*SVdecA: deceleration array of S-curve part in unit of pps

@ ***Return Code***

ERR_NoError
ERR_MoveError

7.9 Linear Interpolated Motion

@ ***Name***

_8134_start_tr_move_xy – Start a relative 2-axis linear interpolation for X & Y, with trapezoidal profile,

_8134_start_ta_move_xy –Start an absolute 2-axis linear interpolation for X & Y, with trapezoidal profile,

_8134_start_sr_move_xy –Start a relative 2-axis linear interpolation for X & Y, with S-curve profile,

_8134_start_sa_move_xy –Start an absolute 2-axis linear interpolation for X & Y, with S-curve profile,

_8134_start_tr_move_zu –Start a relative 2-axis linear interpolation for Z & U, with trapezoidal profile,

_8134_start_ta_move_zu –Start an absolute 2-axis linear interpolation for Z & U, with trapezoidal profile,

_8134_start_sr_move_zu –Start a relative 2-axis linear interpolation

for Z & U, with S-curve profile,

_8134_start_sa_move_zu –Start an absolute 2-axis linear interpolation for Z & U, with S-curve profile,

@ Description

_8134_start_move_%%_xy, _8134_start_move_%%_zu:

These functions cause a linear interpolation motion between two axes without waiting for completion. After issuing this function, it will start to move and leave the function at the same time. Note that xy means the first two axes of one card and zu means the last two axes of one card. %% means speed profile combinations.

@ Syntax

C/C++ (DOS, Windows)

I16 _8134_start_tr_move_xy(I16 CardNo, F64 DistX, F64 DistY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);

I16 _8134_start_ta_move_xy(I16 CardNo, F64 PosX, F64 PosY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);

I16 _8134_start_sr_move_xy(I16 CardNo, F64 DistX, F64 DistY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);

I16 _8134_start_sa_move_xy(I16 CardNo, F64 PosX, F64 PosY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);

I16 _8134_start_tr_move_zu(I16 CardNo, F64 DistX, F64 DistY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);

I16 _8134_start_ta_move_zu(I16 CardNo, F64 PosX, F64 PosY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);

I16 _8134_start_sr_move_zu(I16 CardNo, F64 DistX, F64 DistY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);

I16 _8134_start_sa_move_zu(I16 CardNo, F64 PosX, F64 PosY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);

Visual Basic (Windows)

B_8134_start_tr_move_xy (ByVal CardNo As Integer, ByVal Dist As Double, ByVal Dist As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8134_start_ta_move_xy (ByVal CardNo As Integer, ByVal Pos As Double, ByVal Pos As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8134_start_sr_move_xy (ByVal CardNo As Integer, ByVal Dist As Double, ByVal Dist As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal SVacc As Double, ByVal SVdec As Double) As Integer

B_8134_start_sa_move_xy (ByVal CardNo As Integer, ByVal Pos As Double, ByVal Pos As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal SVacc As Double, ByVal SVdec As Double) As Integer

B_8134_start_tr_move_zu (ByVal CardNo As Integer, ByVal Dist As Double, ByVal Dist As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8134_start_ta_move_zu (ByVal CardNo As Integer, ByVal Pos As Double, ByVal Pos As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer

B_8134_start_sr_move_zu (ByVal CardNo As Integer, ByVal Dist As Double, ByVal Dist As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal SVacc As Double, ByVal SVdec As Double) As Integer

B_8134_start_sa_move_zu (ByVal CardNo As Integer, ByVal Pos As Double, ByVal Pos As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double, ByVal SVacc As Double, ByVal SVdec As Double) As Integer

@ ***Argument***

cardNo: card number designated to perform interpolating function.
Pos: Targer position of one axis
Dist: Target Distance of one axis
StrVel: starting velocity of a velocity profile in unit of pulse per second
MaxVel: starting velocity of a velocity profile in unit of pulse per second
Tacc: specified acceleration time in unit of second
Tdec: specified deceleration time in unit of second
SVacc: specified S-curve acceleration range in unit of pps, default is 0
SVdec: specified S-curve deceleration range in unit of pps, default is 0

@ ***Return Code***

ERR_NoError

7.10 Home Return

@ ***Name***

_8134_set_home_config – Set the configuration for home return.
_8134_home_move – Perform a home return move.

_8134_set_org_offset – Set ORG signal's range
_8134_set_org_logic – Set ORG logic
_8134_set_bounce_filter – Set a bounce filter for homing
_8134_set_org_latch – Enable/diable ORG latch function

@ Description

_8134_set_home_config:

Configure the logic of origin switch and index signal needed for home_move() function. If you need to stop the axis after EZ signal is active(home_mode=1 or 2), you should keep placing ORG signal in the ON status until the axis stop. If the pulse width of ORG signal is too short to keep it at ON status till EZ goes ON, you should select the org_latch as enable. The latched condition is cancelled by the next start or by disabling the org_latch. Three home return modes are available. Refer to Chapter4.1.5 for the setting of home_mode control.

_8134_home_move:

This function will cause the axis to perform a home return move according to the setting of **_8134_set_home_config()** function. The direction of moving is determined by the sign of velocity parameter(svel, mvel). Since the stopping condition of this function is determined by *home_mode* setting, user should take care to select the initial moving direction. Or user should take care to handle the condition when limit switch is touched or other conditions that is possible causing the axis to stop. Executing stop function during **home_move()** can also cause the axis to stop.

_8134_set_org_offset:

This function is used for setting the ORG length. This parameter is active when using home mode 4~7. It will escape ORG during some phases. Please refer to Appendix A.

_8134_set_org_logic:

This function is used for setting the ORG logic. Make sure the ORG logic is correct before homing.

_8134_set_bounce_filter:

This function is used for extend the ORG checking time to prevent mechanical input device's bouncing problem. This parameter is only useful on home mode 4~7. The meaning of the parameter is the times of I/O checking.

_8134_set_org_latch:

This function is used for enable and disable ORG latch function.

@ Syntax

C/C++ (DOS, Windows)

I16 _8134_set_home_config(I16 axis, I16 home_mode, I16 org_logic, I16 ez_logic, I16 ez_count, I16 erc_out)
I16 _8134_home_move(I16 axis, F64 svel, F64 mvel, F64 accel)
I16 _8134_set_org_offset(I16 axis, I16 org_latch)
I16 _8134_set_org_logic(I16 axis, I16 org_logic)
I16 _8134_set_bounce_filter(I16 axis, I16 b_value)

I16 _8134_set_org_latch(I16 axis, I16 org_latch)

Visual Basic (Windows)

B_8134_set_home_config (ByVal axis As Integer, ByVal home_mode As Integer, ByVal org_logic As Integer, ByVal ez_logic As Integer, ByVal ez_count As Integer, ByVal erc_out As Integer) As Integer

B_8134_home_move (ByVal axis As Integer, ByVal str_vel As Double, ByVal max_vel As Double, ByVal accel As Double) As Integer

B_8134_set_org_offset(Byval axis As Integer, ByVal org_latch As Integer) As Integer

B_8134_set_org_logic(ByVal axis As Integer, ByVal org_logic As Integer) As Integer

B_8134_set_bounce_filter(ByVal axis As Integer, ByVal b_value As Integer) As Integer

B_8134_set_org_latch(ByVal axis As Integer, ByVal org_latch As Integer) As Integer

@ ***Argument***

axis: axis number designated to configure and perform home returning

home_mode: stopping modes for home return.
home_mode=0, ORG active only.
home_mode=1, ORG active and then EZ active to stop.
home_mode=2, ORG active and then EZ active slow down to stop.
home_mode=3~7, please refer to the appendix A

org_logic: Action logic configuration for ORG signal
org_logic=0, active low; org_logic=1, active high

org_latch: Latch state control for ORG signal
org_latch=0, don't latch input; org_latch=1, latch input.

EZ_logic: Action logic configuration for EZ signal
EZ_logic=0, active low; EZ_logic=1, active high.

ez_count: 0~15, 0 means count 1 time

erc_out: 0=disable ERC

b_value: Times of I/O checking

org_latch: 0=don't latch, 1=latch

@ ***Return Code***

ERR_NoError

7.11 Manual Pulser Motion

@ ***Name***

_8134_set_pulser_iptmode – Set pulser input mode and operation mode

_8134_pulser_vmove – Begin a manual pulser movement

_8134_set_pulser_ratio – Set manual pulser ratio

_8134_set_step_unit – Set manual pulser ratio

@ *Description*

_8134_set_pulser_iptmode:

Four types of pulse input modes can be available for pulser or hand wheel. User can also move two axes simultaneously with one pulser by selecting the operation mode to ***common mode***. Or move the axes independently by selecting the operation mode to ***independent mode***.

_8134_pulser_vmove:

Begin to move the axis according to manual pulser input as this command is written. The maximum moving velocity is limited by **Speedlimit** parameter. Not until the stop command is written won't system end the manual move mode.

_8134_set_pulser_ratio:

Set the unit number of output pulses to one manual input pulse. The setting range is 0 to 15. If you set n, the unit number is (n+1). That is when one manual pulse input from PA and PB pins, OUT and DIR pins will output n+1 pulses at most. If users do not set the pulser ratio, the default ratio is 0.

_8134_set_step_unit:

Set the unit number of output pulses to one manual input pulse. The setting range is 0 to 15. If you set n, the unit number is (n+1). That is when one manual pulse input from PA and PB pins, OUT and DIR pins will output n+1 pulses at most. If users do not set the pulser ratio, the default ratio is 0.

@ *Syntax*

C/C++ (DOS, Windows)

I16 _8134_set_pulser_iptmode(I16 axis, I16 Inputmode, I16 Indep_com)

I16 _8134_pulser_vmove(I16 axis, F64 Speedlimit)

I16 _8134_set_pulser_ratio(I16 AxisNo, I16 Value)

I16 _8134_set_step_unit(I16 AxisNo, I16 unit)

Visual Basic (Windows)

B_8134_set_pulser_iptmode (ByVal axis As Integer, ByVal Inputmode As Integer, ByVal Indep_com As Integer) As Integer

B_8134_pulser_vmove (ByVal axis As Integer, ByVal Speedlimit *As Double) As Integer*

@ *Argument*

axis: axis number designated to start manual move

Inputmode: setting of manual pulser input mode from PA and PB pins

ipt_mode=0, 1X AB phase type pulse input.
ipt_mode=1, 2X AB phase type pulse input.
ipt_mode=2, 4X AB phase type pulse input.
ipt_mode=3, CW/CCW type pulse input.

Indep_com: common or independent mode selection

op_mode=0, Independent for each axis
op_mode=1,PAX, PBX common for PAY, PBY

or PAZ, PBZ common for PAU, PBU.

Speedlimit: limitation for maximum velocity

Value: pulser ratio, its value n should satisfy the following relations,

PA & PB Input Mode	Applicable Range
1 times multiplied 90° phase difference signal	fP<fH/(n+1)
2 times multiplied 90° phase difference signal	fP<fH/[(n+1)×2]
4 times multiplied 90° phase difference signal	fP<fH/[(n+1)×4]
2-pulse input	fP<fH/(n+1)

where fP is the maximum input frequency (pps) of pulser signals and fH is the frequency (pps) of the output signals. If the pulser ratio is not set by these rules, the output pulses will not appear in the cycle of one manul input pulse or in the rising edge of the output pulses.
unit: pulser ratio

@ ***Return Code***

ERR_NoError

@ ***Coding Example in C Language***

1. Satisfication of fP<fH/[(n+1)×m]

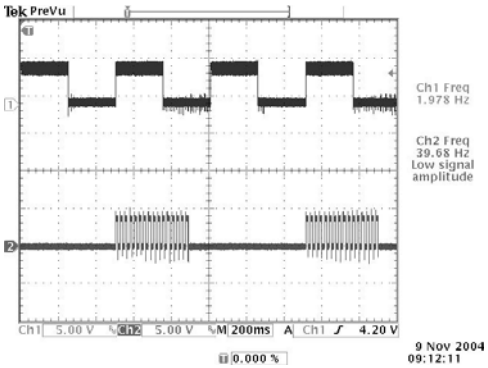
```
_8134_set_pulser_iptmode(0,1,0); // set pulser input mode
_8134_set_pulser_ratio(0,5); // set pulser ratio
_8134_pulser_vmove(0,40);
```

8

From the above figure, Ch1 is the input signal of the pulsers, and Ch2 is the relative output signal from OUT and DIR pins. Obviously, the output signal frequency satisfies the relationship,
FP(=1.978Hz) < fH(=40Hz)/[(5+1)×2].

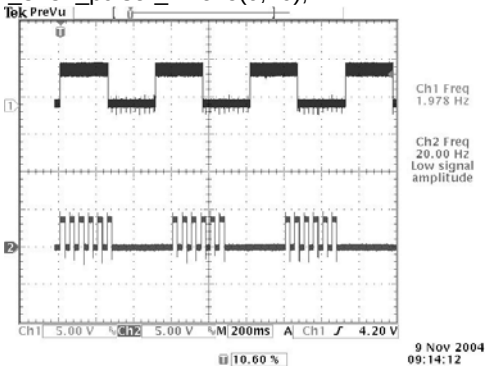
2. Dissatisfication of fP<fH/[(n+1)×m]

```
_8134_set_pulser_iptmode(0,1,0); // set pulser input mode
_8134_set_pulser_ratio(0,15); // set pulser ratio
_8134_pulser_vmove(0,40);
```



3. Dissatisfication of fP<fH/[(n+1)×m]

```
_8134_set_pulser_iptmode(0,1,0); // set pulser input mode
_8134_set_pulser_ratio(0,5); // set pulser ratio
_8134_pulser_vmove(0,20);
```



7.12 Motion Status

@ Name

_8134_motion_done – Return the status when a motion is done

@ Description

_8134_motion_done:

Return the motion status of PCI-8134. position.
Definition of return value is as following:

Return value =

0: Motion Stop.
1: Waiting STA
2: Waiting INP
3: In Accelerating
4: In Target Speed

5: In Decelerating

6: In Start Speed
7: Waiting other axes

The following code demonstrates how to utilize this function:

```
                    // Begin a trapezoidal velocity profile motion
_8134_start_ta_move(axis_x, pos1, svel, mvel, Tacc,Tdec);

// Wait for completion
while(motion_done(axis_x) !=0) ;
```

If the axis is running under home mode 4~7, this function will return the homing phase. Please refer to Appendix A for details.

@ ***Syntax***

C/C++ (DOS, Windows)

I16 _8134_motion_done(I16 axis)

Visual Basic (Windows)

B_8134_motion_done (ByVal axis As Integer) As Integer

@ ***Argument***

axis: axis number of motion status

@ ***Return Code***

ERR_NoError

7.13 Servo Drive Interface

@ ***Name***

_8134_set_alm – Set alarm logic and alarm mode
_8134_set_inp – Set In-Position logic and enable/disable
_8134_set_erc_enable – Set ERC pin output enable/disable

@ ***Description***

_8134_set_alm:

Set the active logic of **ALARM** signal input from servo driver. Two reacting modes are available when **ALARM** signal is active.

_8134_set_inp:

Set the active logic of **In-Position** signal input from servo driver. Users can select whether they want to enable this function. Default state is disabled.

_8134_set_erc_enable:

You can set ERC pin output enable/disable by this function. Default state is enabled.

@ ***Syntax***

C/C++ (DOS, Windows)

I16 set_alm(I16 axis, I16 alm_logic, I16 alm_mode)

I16 set_inp(I16 axis, I16 inp_enable, I16 inp_logic)
I16 set_erc_enable(I16 axis, I16 erc_enable)

Visual Basic (Windows)

B_8134_set_alm_logic (ByVal axis As Integer, ByVal alm_logic As Integer, ByVal alm_mode As Integer) As Integer
B_8134_set_inp_logic (ByVal axis As Integer, ByVal inp_enable As Integer, ByVal inp_logic As Integer) As Integer
B_8134_set_erc_enable(ByVal axis As Integer, ByVal erc_enable As Integer) As Integer

@ Argument

axis: axis number designated to configure
alm_logic: setting of active logic for ALARM signal
alm_logic=0, active LOW.
alm_logic=1, active HIGH.
inp_logic: setting of active logic for INP signal
inp_logic=0, active LOW.
inp_logic=1, active HIGH.
alm_mode: reacting modes when receiving ALARM signal.
alm_mode=0, motor immediately stops.
alm_mode=1, motor decelerates then stops.
inp_enable: INP function enable/disable
inp_enable=0, Disabled
inp_enable=1, Enabled
erc_enable: ERC pin output enable/disable
erc_enable=0, Disabled
erc_enable=1, Enabled

@ Return Code

ERR_NoError

7.14 I/O Control and Monitoring

@ Name

_8134_set_servo – Set the output pin for servo ON control
_8134_get_io_status – Get all the I/O status

@ Description

_8134_set_servo:

Set the High/Low output state of general purpose output pin **SVON**. Ususally, users can connect this pin to AC servo's Servo ON pin.

_8134_get_io_status:

Get all the I/O status for each axis. The definition for each bit is as following:

Bit	Name	Description
0	+EL	Positive Limit Switch
1	-EL	Negative Limit Switch
2	+SD	Positive Slow Down Point
3	-SD	Negative Slow Down Point
4	ORG	Origin Switch
5	EZ	Index signal
6	ALM	Alarm Signal
7	SVON	SVON of PCL5023 pin output
8	RDY	RDY pin input
9	INT	Interrupt status
10	ERC	ERC pin output
11	INP	In-Position signal input

@ ***Syntax***

C/C++ (DOS,Windows)

I16 _8134_set_servo(I16 axis, I16 on_off)

*I16 _8134_get_io_status(I16 axis, U16 *io_status)*

Visual Basic (Windows)

B_8134_set_servo (ByVal axis As Integer, ByVal on_off As Integer) As Integer

B_8134_get_io_status (ByVal axis As Integer, io_sts As Integer) As Integer

@ ***Argument***

axis: axis number for I/O control and monitoring

on_off: setting for SVON pin digital output
on_off=0, SVON is LOW.
on_off=1, SVON is HIGH.

***io_status**: I/O status word. Where "1' is ON and "0" is OFF. ON/OFF state is read based on the corresponding set logic.

@ ***Return Code***

ERR_NoError

7.15 Position Counter Control

@ ***Name***

_8134_set_position – Set the actual position

_8134_get_position – Get the actual position

_8134_set_command – Set the current command target value

_8134_get_command – Get the current command target value

_8134_get_error_counter – Get current error counter value

_8134_reset_error_counter – Reset error counter value

_8134_set_feedback_error_detect – Set feedback error detect value

@ Description

_8134_set_position:

Changes the current actual position to the specified position.

_8134_get_position:

Reads the current actual position. Note that when feedback signals is not available in the system, thus external encoder feedback is *Disabled* in **_**8134**_set_feedback_src()** function, the value gotten from this function is command position.

_8134_set_command:

Changes the command position to the specified command position. The command position is the target position of this command, not the current command position counter's value.

_8134_get_command:

Reads the current command position.

The command position is the target position of this command, not the current command position counter's value.

_8134_get_error_counter:

Read the error counter value which is calculated from command and feedback counter.

_8134_reset_error_counter:

Reset the error counter value to 0.

_8134_set_feedback_error_detect:

Set the error counter detect value. if the error counter is greater than this value, the out-of-step interrupt will be issued.

@ Syntax

C/C++ (DOS, Windows)

I16 _8134_set_position(I16 axis, F64 pos)
*I16 _8134_get_position(I16 axis, F64 *pos)*

I16 _8134_set_command(I16 axis, F64 cmd)
*I16 _8134_get_command(I16 axis, F64 *cmd)*
*I16 _8134_get_error_counter(I16 axis, I16 *error_c)*
I16 _8134_reset_error_counter(I16 axis)
I16 _8134_set_error_feedback_detect(I16 axis, I32 max_error)

Visual Basic (Windows)

B_8134_get_position (ByVal axis As Integer, pos As Double) As Integer
B_8134_set_position (ByVal axis As Integer, ByVal pos As Double) As Integer
B_8134_get_command (ByVal axis As Integer, cmd As Double) As Integer
B_8134_set_command (ByVal axis As Integer, ByVal cmd As Double) As Integer
B_8134_get_error_counter(ByVal axis As Integer, error_c As Integer) As Integer
B_8134_reset_error_counter(ByVal axis As Integer) As Integer
B _8134_set_error_feedback_detect(ByVal axis As Integer, ByVal

max_error As Long) As Integer

@ ***Argument***

axis: axis number designated to set and get position.
pos: actual position or command position
cmd: target command position value
***error_c**: error counter value
max_error: error detect value setting

@ ***Return Code***

ERR_NoError

7.16 Interrupt Control

@ ***Name***

_8134_int_enable – Set interrupt event enable
_8134_int_disable – Set interrupt event disable
_8134_int_control – Enable/Disable IRQ channel
_8134_set_int_factor – Set interrupt generating factors
_8134_get_int_status – Get the interrupting status of axis
_8134_link_axis_interrupt – Link a interrupt callback function

@ ***Description***

_8134_int_enable:

This function is used to enable interrupt event generating to host PC. **(Window only).**

_8134_int_disable:

This function is used to disable interrupt event generating to host PC. **(Window only).**

_8134_int_control :

This function is used to control the hardware interrupt channel enable or disable. Please call this function after the interrupt events are enabled.

_8134_set_int_factor:

This function allows users to select factors to initiate the INT signal. PCI-8134 can generate INT signal to host PC by setting the relative bit as 1. The definition for each bit is as following:

Bit	Interrupt Factor
0	Stop with the EL signal
1	Stop with the SD signal
2	Stop with the ALM signal
3	Stop with the STP signal
4	Should be set to 0
5	Completion of home return
6	Completion of preset movement (PTP move)

7	Completion of interpolating motion for two axes (X & Y) or (Z & U)
8~12	Should be set to 0
13	When ***stop*** function stop the axis
14	EA/EB, PA/PB encoder input error
15	start with STA signal
16	Completion of acceleration
17	Start of deceleration
18~22	Should be Set to 0
23	RDY active (AP3 of PCL5023 change from 1 to 0)
24~31	Should be set to 0

Note: Bit 14: The interrupt is generated when pins EA and EB, or PA and PB change simultaneously. It means there has an encoder input error.

_8134_get_int_axis:

This function allows user to identify which axis generates the INT signal to host PC. (**DOS only**)

_8134_get_int_status:

This function allows user to identify what kinds of interrupt is generated.

After user gets this value, the status register will be cleared to 0. The return value is a 32 bits unsigned integer and the definition for each bit is as following:

Bit	Interrupt Type
0	Stop with the +EL signal
1	Stop with the –EL signal
2	Stop with the +SD signal
3	Stop with the –SD signal
4	Stop with the ALM signal
5	Stop with the STP signal
6	Always 0
7	Always 0
8	Stop with v_stop() command
9	Stop with home return completed
10	Always 0
11	Stop with preset move completed (PTP move)
12	Stop with EA/EB input error
13	Always 0
14	Stop with PA/PB input error
15	Start with STA signal
16	Acceleration Completed
17	Deceleration Started
18~22	Always 0

23	RDY active (AP3 of PCL5023 change from 1 to 0)
24~31	Always 0

_8134_link_axis_interrupt:

This function is used to create a callback function in Windows for interrupt signal receiving. Once the interrupt comes, the callback function will be called too.

@ *Syntax*

C/C++ (DOS,Windows)

*I16 _8134_get_int_axis(U16 *int_axis) (DOS only)*
*I16 _8134_int_enable (I16 cardNo, HANDLE *phEvent)*
I16 _8134_int_disable (I16 cardNo)
I16 _8134_int_control(I16 cardNo, I16 intFlag)
I16 _8134_set_int_factor(I16 axis, U32 int_factor)
*I16 _8134_get_int_status(I16 axis, U32 *int_status)*
*I16 _8134_link_axis_interrupt(I16 AxisNo, void (_stdcall *callbackAddr)(void))*

Visual Basic (Windows)

B_8134_int_enable (ByVal cardNo As Integer, phEvent As Long)
B_8134_int_disable (ByVal cardNo As Integer) As Integer
B_8134_int_control (ByVal cardno As Integer, ByVal intFlag As Integer)
B_8134_set_int_factor (ByVal axis As Integer, ByVal int_factor As Long) As Integer
B_8134_get_int_status (ByVal axis As Integer, int_status As Long) As Integer

B_8134_link_axis_interrupt(ByVal AxisNo As Integer, By Val callbackAddr as Long) As Integer

@ ***Argument***

cardNo: card number 0,1,2,3…
axis: axis number 0,1,2,3,4…
intFlag: int flag, 0:disable or 1:enable
phEvent: event or event array for interrupt axis (For Windows only)
int_factor: interrupt factor, refer to previous interrupt factor table
int_axis: interrupt axis number (the return value)
int_status: interrupt factor (the return value), refer to previous interrupt type table
callbackAddr: The call back function address

@ ***Return Code***

ERR_NoError

8

Connection Example

This chapter shows some connection examples between PCI-8134/PCI-8134A and servo drivers and stepping drivers.

8.1 General Description of Wiring

Figure 8.1 is a general description of all the connectors of PCI-8134. Only connection of one of 4 axes is shown.

CN1: Receives +24V power from external power supply.

CN2: Main connection between PCI-8134/PCI-8134A and pulse input servo driver or stepping driver.

CN3: Receive pulse command from manual pulser.

CN4: Connector for simultaneously start or stop multiple PCI-8134/PCI-8134A cards.

Figure 8.2 shows how to integrate PCI-8134/PCI-8134A with a physical system.

Description of PCI-8134/PCI-8134A Indexer Pinouts

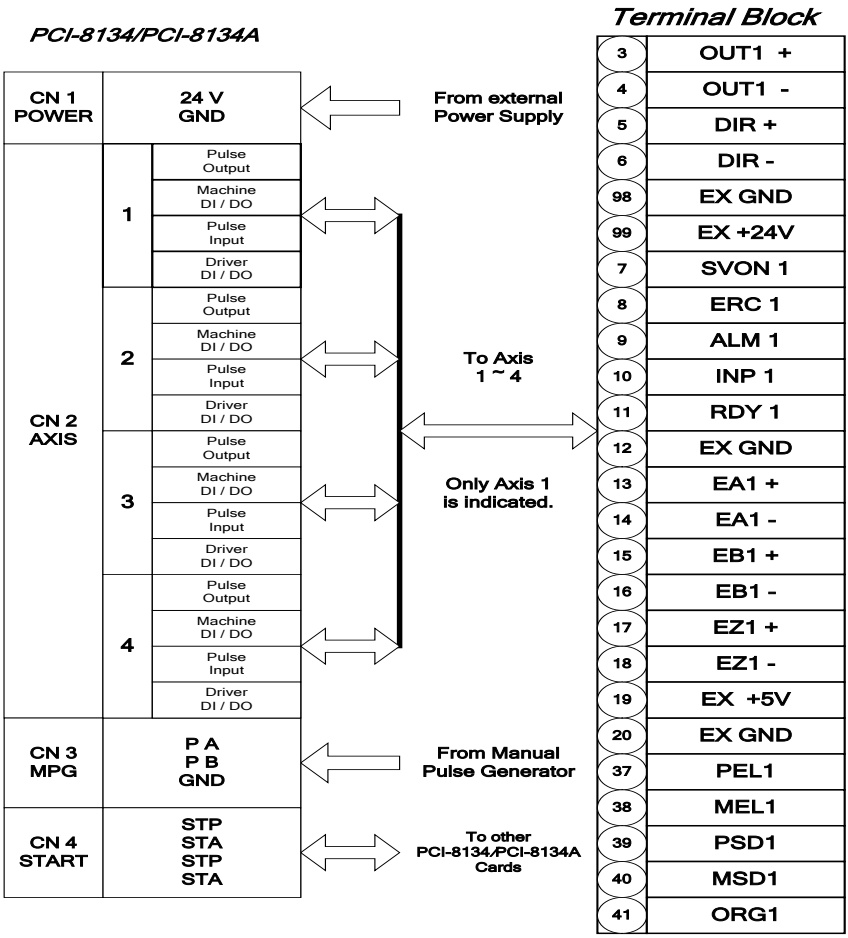


Figure 8.1 General Description of Wiring

Wiring of PCI-8134 with Servo Driver

Wiring of PCI-8134/PCI-8134A with Servo Driver

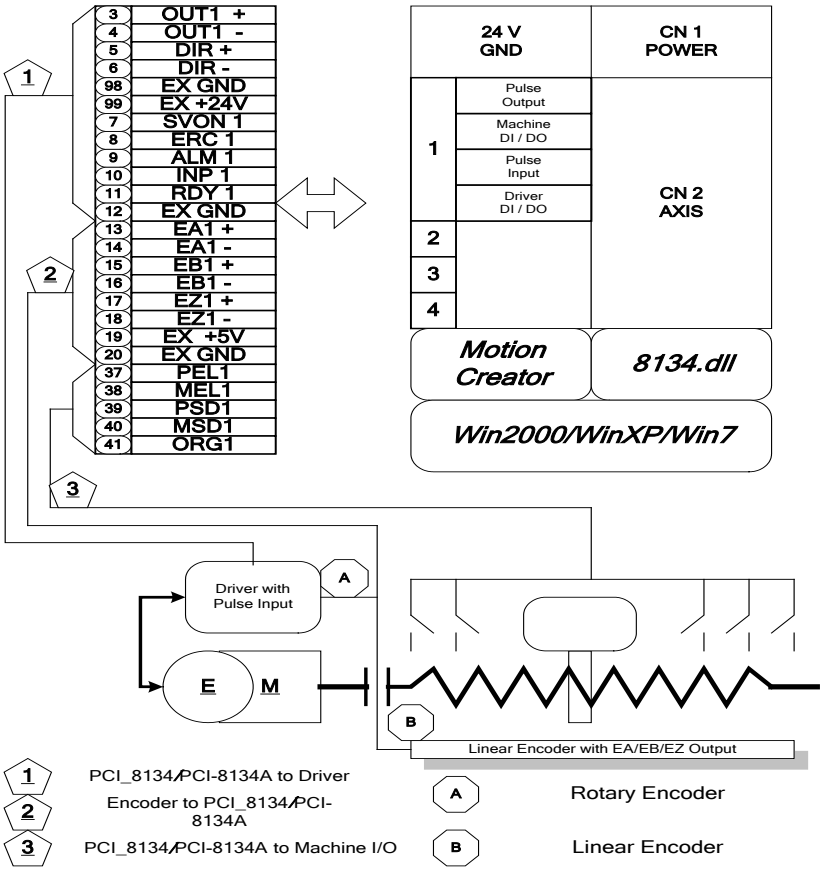


Figure 8.2 System Integration with PCI-8134/PCI-8134A

8.2 Connection Example with Servo Drive

In this section, we use ***Panasonic Servo Drive*** as an example to show how to connect it with ***PCI-8134/PCI-8134A***. Figure 8.3 show the wiring.

Note that:

1. For convenience' sake, the drawing shows connections for one axis only.
2. Default pulse output mode is **OUT/DIR** mode; default input mode is 4X **AB phase** mode. Anyway, user can set to other mode by software function.
3. Since most general purpose servomotor driver can operates in ***Torque Mode; Velocity Mode; Position mode.*** For linking with PCI-8134/PCI-8134A, user should set the operating mode to Position Mode. By setting servo driver to this mode, user can use PCI-8134/PCI-8134A to perform either ***Position Control*** or ***Velocity Control***.
4. The ***Deviation Counter Clear*** input for Panasonic Drive is line drive type where ***ERC*** output of PCI-8134/PCI-8134A is open collector type. So a little circuit is required for interfacing.

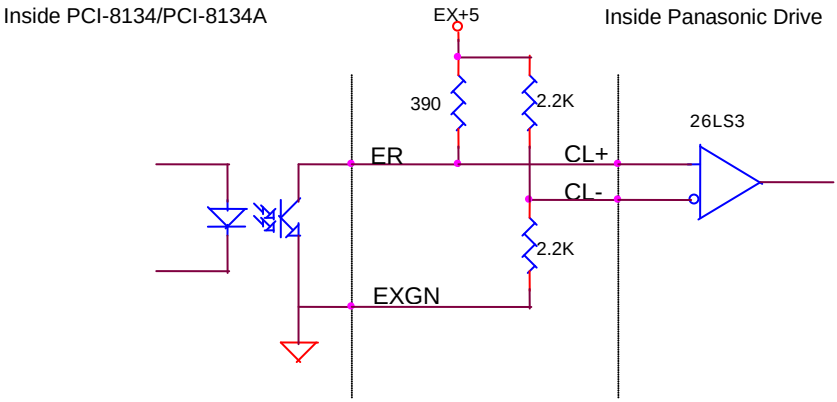


Figure 8.3 Interface circuit between ERC and (CL+, CL-)

Wiring of PCI-8134/PCI-8134A with Panasonic MSD

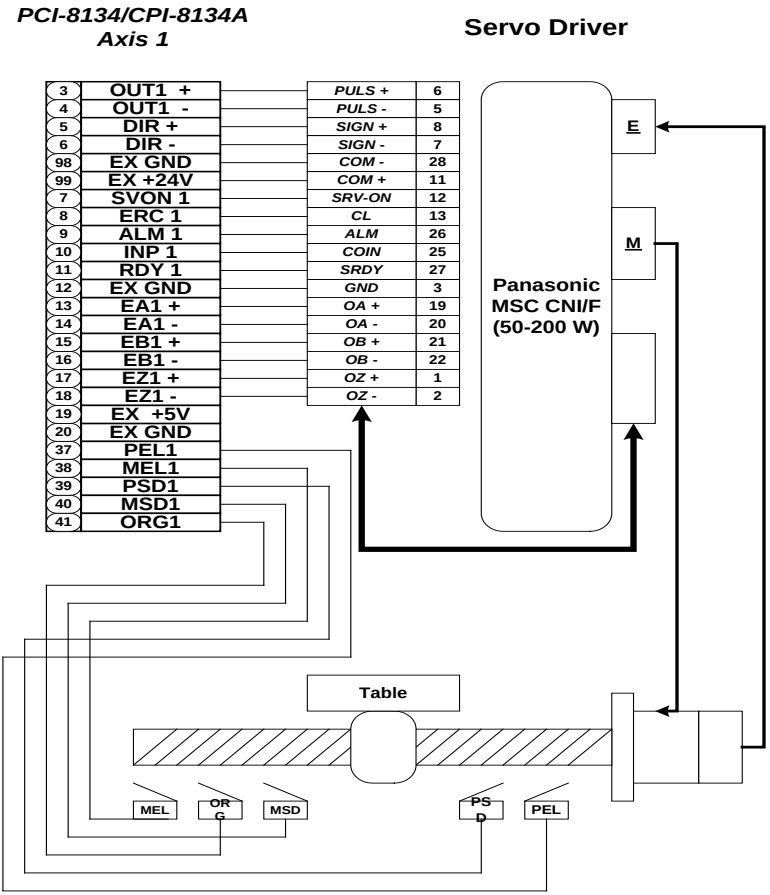


Figure **8.4** Connection of PCI-8134 **with Panasonic Drive**

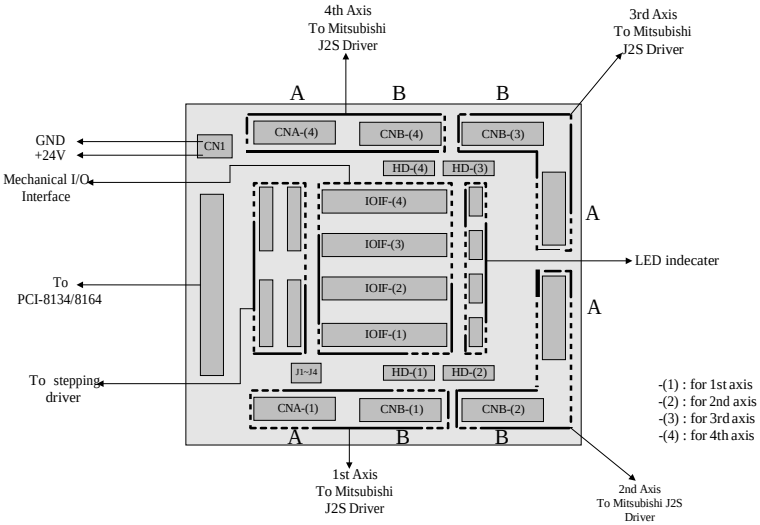
8.3 Wiring with DIN-814M

Warning

The DIN-814M is used for wiring between **Mitsubishi J2S** series servo drivers and ADLINK **PCI-8134/PCI-8134A, PCI-8164, or MPC-8164** motion controller card ONLY.

Note:

1. The DIN-814M provides 2 connection methods for every axis. The first is through the CNA & CNB connectors. This is for Mitsubishi J2S series servo driver. The second is through SJ connector. This is for stepping driver or other servo drivers (for Panasonic MINAS MSD driver, please use DIN-814P). Keep in mind that the signals in SJ and CNA & CNB of

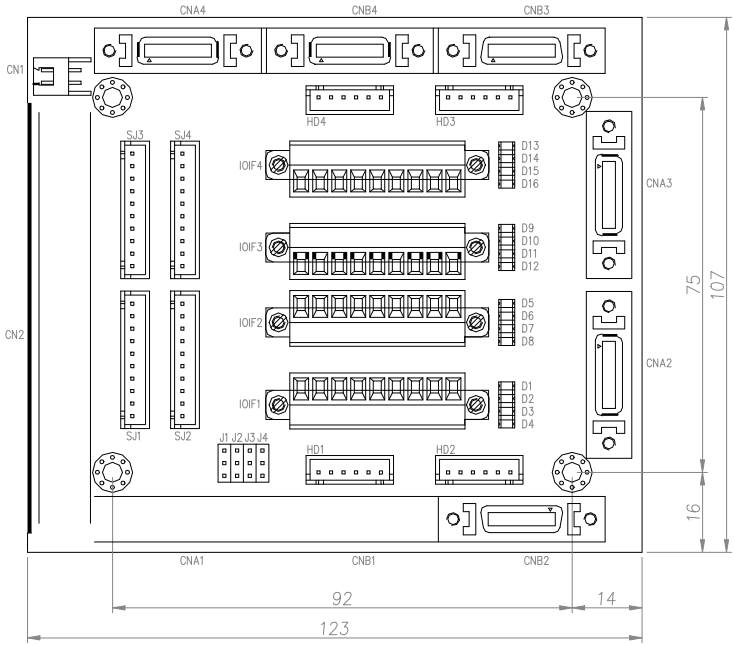


the same axis are directly shorted. DO NOT use both connectors at the same time.

2. Two one-to-one 20-PIN cables are required for connection between the CNA & CNB and the Mitsubishi J2S driver. It is available from ADLINK, or users may contact the local dealer or distributor to get cable information.

3. Depending on which PCI-8134/PCI-8134A or PCI-8164/MPC-8164 card used, some signals (PSD and MSD) in the IOIF connector will function differently. When PCI-8134 is used, The PSD and MSD are for positive slow down and negative slow down signal respectively. While PCI-8164 is used, PSD is for CMP and LTC and MSD is for SD. For more detail s, please refer to the PCI-8134 and PCI-8164 user manuals.
4. Ext EMG and EMG: Due to the existence of EMG (Emergence stop signal) in the Mitsubishi J2S driver, users may select either of the following two operations by setting jumpers (J1-J4, J1 for 1st axis, J2 for 2nd axis, etc.).
    - **1-2 shorted:** The EMG is shorted to GND, so Ext. EMG of IOIF pin 2 is not used.
    - **2-3 shorted:** The Ext. EMG of IOIF pin 2 is connected to EMG at the driver; so, to externally stop the motor set Ext. EMG open to GND.

Mechanical Dimensions:



PIN Assignment:

CNA1~CNA4

No.	Name	I/O	Function	No.	Name	I/O	Function
1	IGND	--	Isolated Ground	2	DIR+	O	Direction Signal (+)
3	OUT+	O	Pulse Signal (+)	4			
5	EZ+	I	Encoder Z-phase (+)	6	EA+	I	Encoder A-phase (+)
7	EB+	I	Encoder B-phase (+)	8	ERC	O	Error counter Clear
9	+24V	O	Voltage output	10	IGND	--	Isolated Ground
11				12	DIR-	O	Direction Signal (-)
13	OUT-	O	Pulse Signal (-)	14			
15	EZ-	I	Encoder Z-phase (-)	16	EA-	I	Encoder A-phase (-)
17	EB-	I	Encoder B-phase (-)	18	INP	I	Servo In Position
19	RDY	I	Servo Ready	20	IGND	--	Isolated Ground

CNB1~CNB4

No.	Name	I/O	Function	No.	Name	I/O	Function
1	IGND	--	Isolated Ground	2			
3				4			
5	Servo ON	O	Servo On	6			
7				8			
9				10	IGND	--	Isolated Ground
11				12			
13	+24V	O	Voltage output	14			
15	EMG	I	Internal EMG Signal	16	IGND	--	Isolated Ground
17	IGND	--	Isolated Ground	18	ALM	I	Servo Alarm
19				20	IGND	--	Isolated Ground

IOIF1~IOIF4

No.	Name	I/O	Function	No.	Name	I/O	Function
1	+24V	O	Voltage output	6	MSD	I	Negative Slow Switch (+)
2	EX_EMG	I	External EMG Signal	7	ORG	I	
3	PEL	I	Positive Limit (+)	8	IGND	--	
4	MEL	I	Negative Limit (-)	9	IGND	--	
5	PSD	I	Positive Slow Switch (+)				

SJ1~SJ4

No.	Name	I/O	Function	No.	Name	I/O	Function
1	OUT+	O	Pulse Signal (+)	6	ALM	I	Servo Alarm
2	OUT-	O	Pulse Signal (-)	7	+5V	O	Voltage output
3	DIR+	O	Direction Signal (+)	8	Servo ON	O	Servo On
4	DIR-	O	Direction Signal (-)	9	+5V	O	Voltage output
5	EZ+	I	Index Signal	10	IGND	--	Isolated Ground

CN1

No.	Name	I/O	Function
1	EX+24V	I	External Power Supply Input (+24V DC ± 5%)
2	EXGND	--	External Power Supply Ground.

HD1~HD4

No.	Name	I/O	Function	No.	Name	I/O	Function
1	+24V	O	Voltage output	4	EX_EMG	I	External EMG Signal

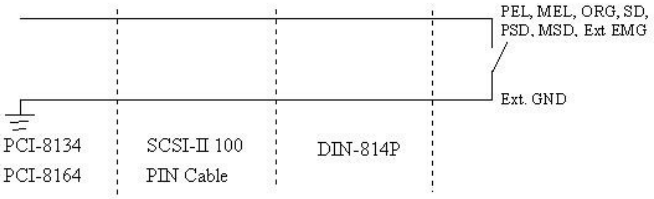
2	Servo ON	O	Servo On	5	ALM	I	Servo Alarm
3	RDY	I	Servo Ready	6	IGND	--	Isolated Ground

Jumper

J1~J4	1: GND	2: EMG4	3: EX_EMG	

How to wire

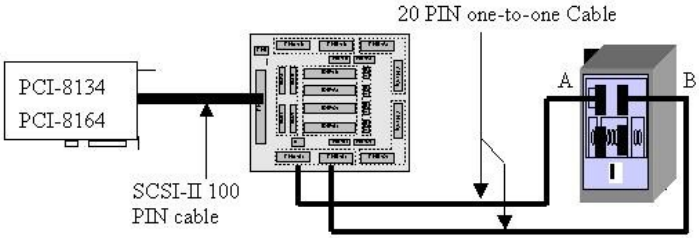
PEL, MEL, ORG, SD, PSD, MSD, Ext.EMG (in IOIF):



CMP, LTC (in IOIF)

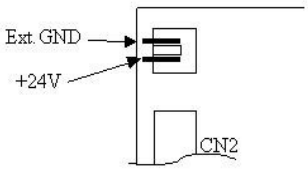
CMP is a TTL 5V or 0V output (vs. Ext GND) LTC is a TTL 5V or 0V input (vs. Ext. GND)

CNA & CNB, CN2



SJ: Please refer to PCI-8134/PCI-8134A/PCI-8164 user manual for wiring.

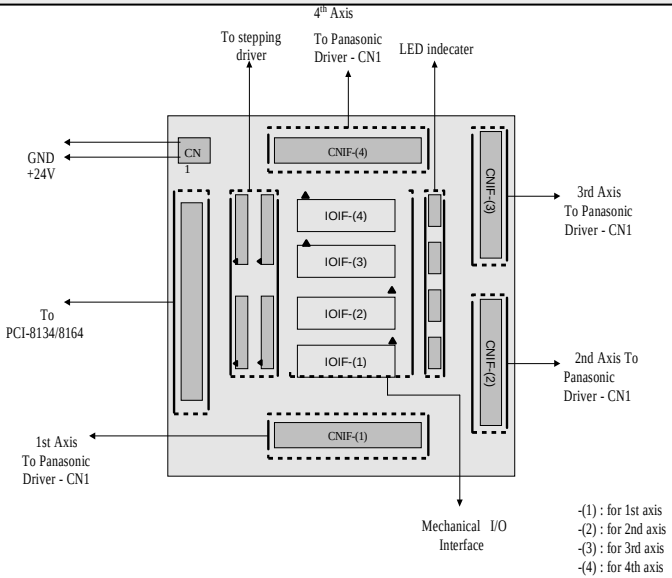
CN1:



8.4 Wiring with DIN-814P

Warning

The DIN-814M is used for wiring between the **Panasonic MINAS MSD** series servo driver and ADLINK **PCI-8134/PCI-8134A, PCI-8164** motion controller cards

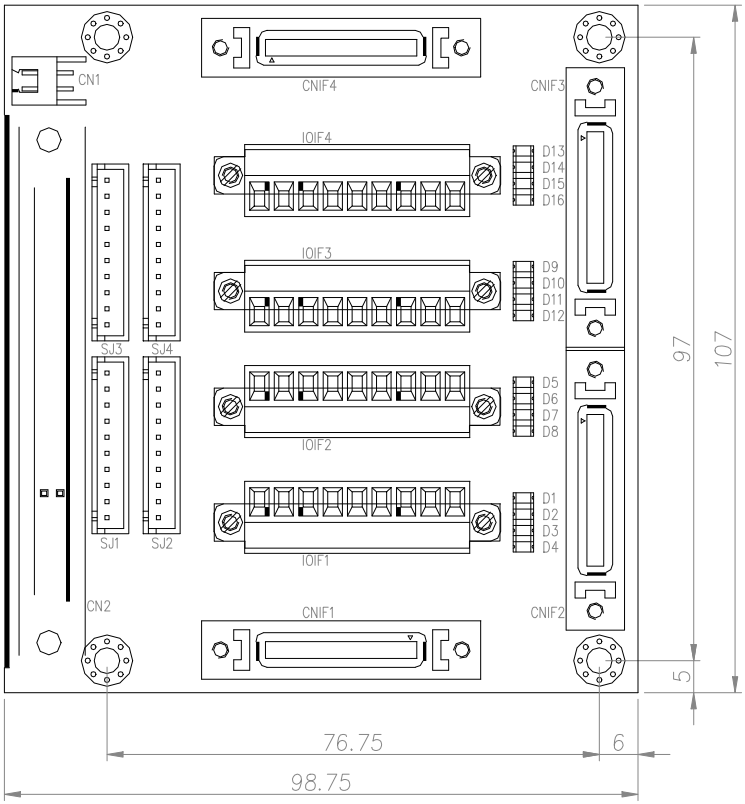


Note:

1. The DIN-814P provides 2 connection methods for every axis. The first is through the CNIF connector for the Panasonic MINAS MSD series servo driver. The second is through SJ connector for stepping drivers or other servo drivers (for the Mitsubishi J2S driver, please use DIN-814M). Keep in mind that the signals in SJ and CNIF of the same axis are directly shorted. DO NOT use both connectors at the same time.
2. A one-to-one 36-PIN cable is required to connect between the CNIF and the Panasonic MINAS MSD driver. It is available from ADLINK, or users may contact a local dealer or distributor to get cable information.

3. Depending on the PCI-8134/PCI-8134A or PCI-8164 card used, some signals (PSD & MSD) in the IOIF connector will function differently. When PCI-8134/PCI-8134A is used, the PSD and MSD signals are for positive slow down and negative slow down signal respectively. When PCI-8164 is used, PSD is for CMP and LTC, and MSD is for SD. For more details, please refer to the PCI-8134/PCI-8134A and PCI-8164 user manuals.

Mechanical Dimensions:



PIN Assignment:

CNIF1~CNIF4

No.	Name	I/O	Function	No.	Name	I/O	Function
1	EZ+	I	Encoder Z-phase (+)	2	EZ-	I	Encoder Z-phase (-)
3	IGND	--	Isolated Ground	4			
5	OUT+	O	Pulse Signal (+)	6	OUT-	O	Pulse Signal (-)
7	DIR+	O	Direction Signal (+)	8	DIR-	O	Direction Signal (-)
9	IGND	--	Isolated Ground	10			
11	+24V	O	Voltage output	12	Servo ON	O	Servo On
13	ERC	O	Error counter Clear	14			
15	IGND	--	Isolated Ground	16			
17				18			
19	EA+	I	Encoder A-phase (+)	20	EA-	I	Encoder A-phase (-)
21	EB+	I	Encoder B-phase (+)	22	EB-	I	Encoder B-phase (-)
23				24			
25	INP	I	Servo In Position	26	ALM	I	Servo Alarm
27	RDY	I	Servo Ready	28	IGND	--	Isolated Ground
29				30			
31				32			
33				34			
35				36			

IOIF1~IOIF4

No.	Name	I/O	Function	No.	Name	I/O	Function
1	+24V	O	Voltage output	6	MSD	I	Negative Slow Switch (+)
2	+24V	O	Voltage output	7	ORG	I	
3	PEL	I	Positive Limit (+)	8	IGND	--	
4	MEL	I	Negative Limit (-)	9	IGND	--	
5	PSD	I	Positive Slow Switch (+)				

SJ1~SJ4

No.	Name	I/O	Function	No	Name	I/O	Function
1	OUT+	O	Pulse Signal (+)	6	ALM	I	Servo Alarm
2	OUT-	O	Pulse Signal (-)	7	+5V	O	Voltage output
3	DIR+	O	Direction Signal (+)	8	Servo ON	O	Servo On
4	DIR-	O	Direction Signal (-)	9	+5V	O	Voltage output
5	EZ+	I	Index Signal	10	IGND	--	Isolated Ground

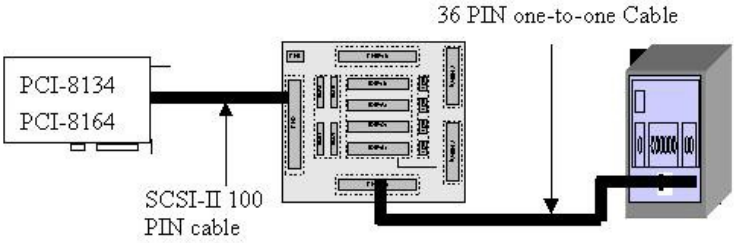
CN1

No.	Name	I/O	Function
1	EX+24V	I	External Power Supply Input (+24V DC ± 5%)
2	EXGND	--	External Power Supply Ground

CMP, LTC (in IOIF)

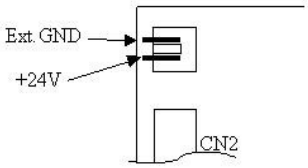
CMP is a TTL 5V or 0V output (vs. Ext GND) LTC is a TTL 5V or 0V input (vs. Ext. GND)

CNA & CNB, CN2



SJ: Please refer to PCI-8134/PCI-8134A/PCI-8164 user manual for wiring.

CN1:



8.5 Wiring with DIN-814PA

DIN-814PA is a termination board for Panasonic MINAS A series servo drivers. The connectors are 50-pins

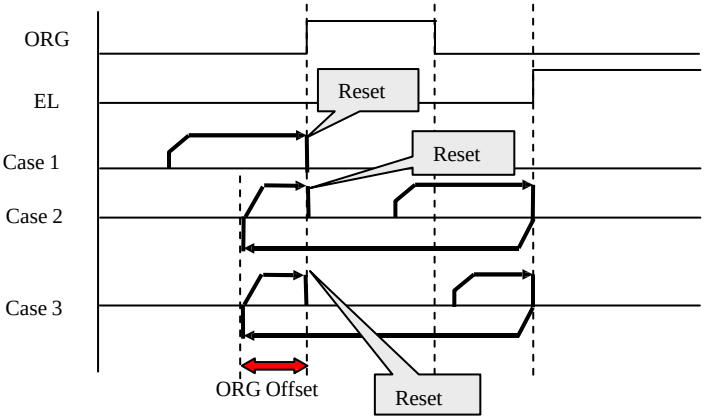
Appendix A: Auto Home Return Modes

PCI-8134/PCI-8134A provides 5 extra home return modes from 3 to 7 which are implemented in an independent thread under Win32 system. These modes are based on original 3 basic modes and can achieve auto searching jobs during homing. Users do not worry about the end-limit case and starting position when homing.

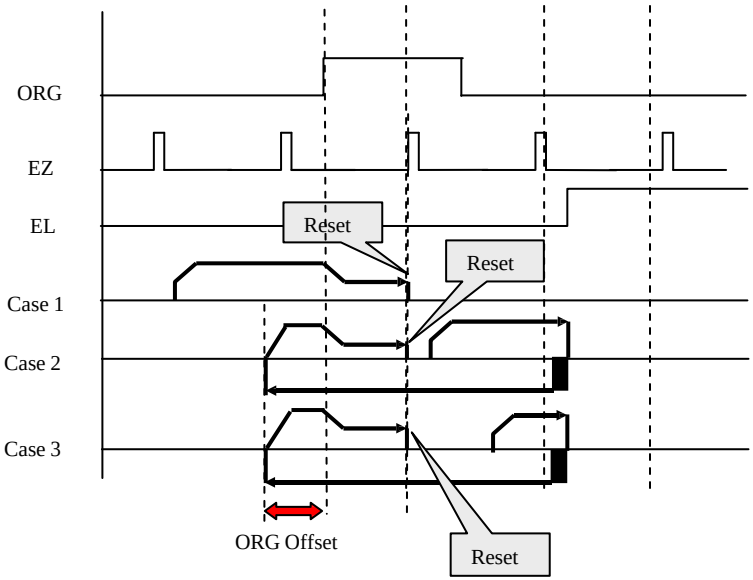
The following figures show the timing charts of these extra home modes. Users should take notice of the timing charts and be aware of the limitations.

home_mode = 3 : Based on home mode 0

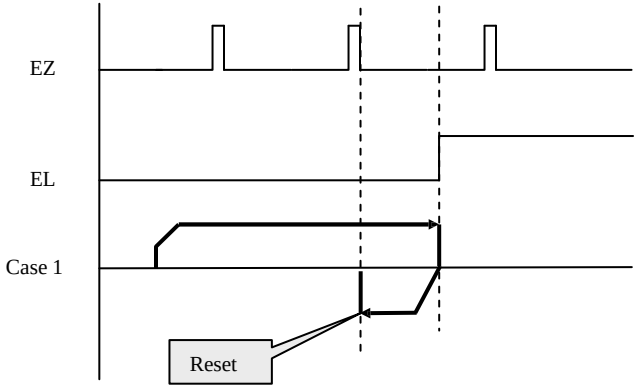
It can search the ORG edge signal at a specific direction automatically. _8134_motion_done() function will return a phase code during homing.



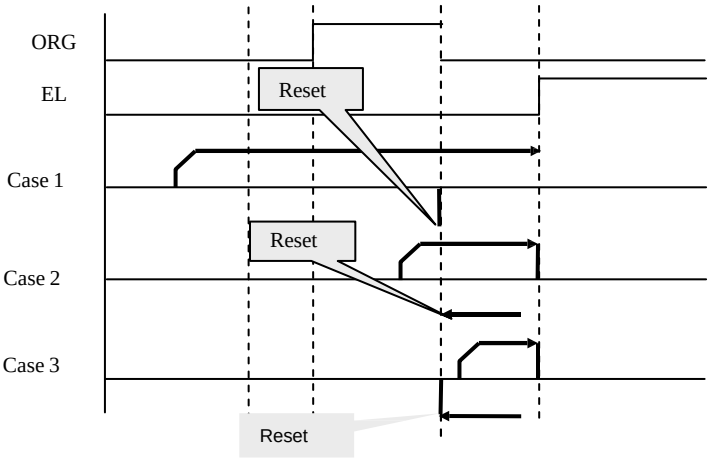
home_mode = 4 : Based on home mode 2



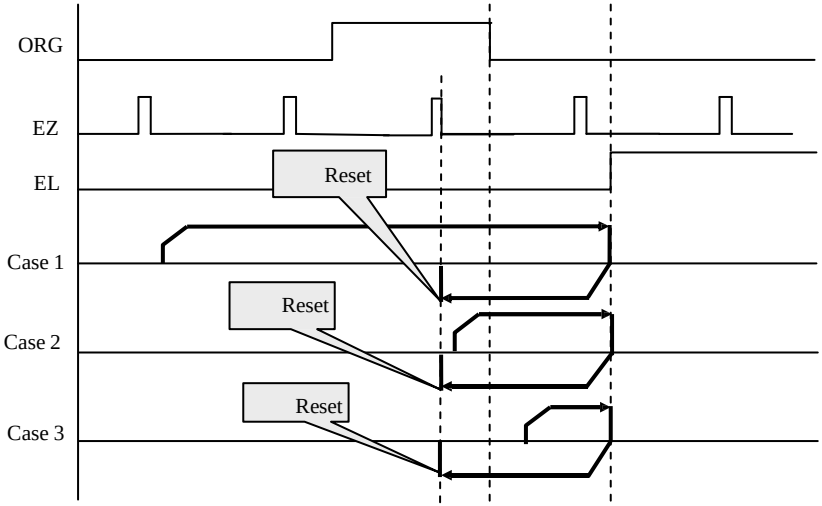
home_mode = 5 : Based on home mode 1



home_mode = 6 : Based on home mode 0

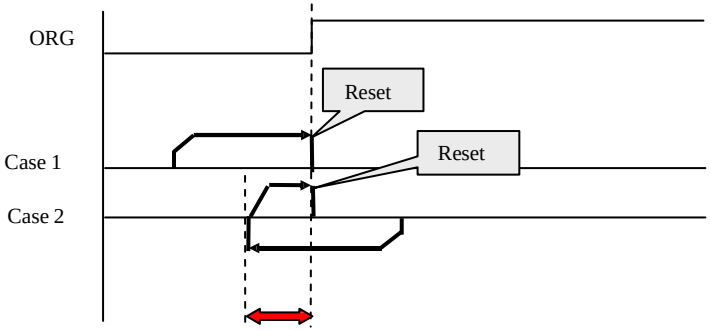


home_mode = 7 : Based on home mode 2



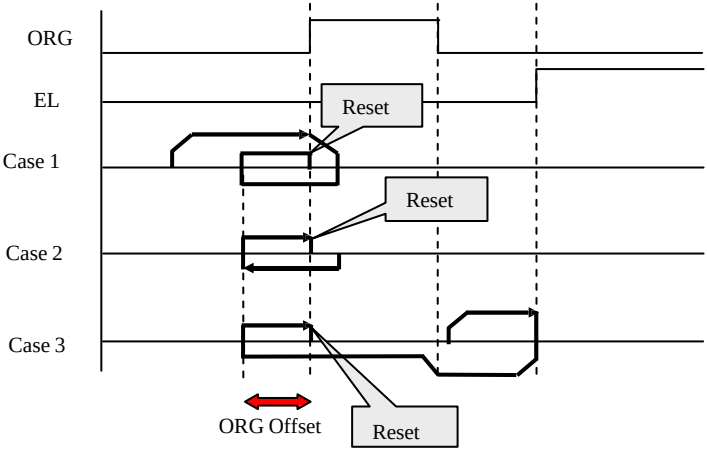
home_mode = 8 : Based on home mode 0

No limit switches and ORG is always ON at one direction

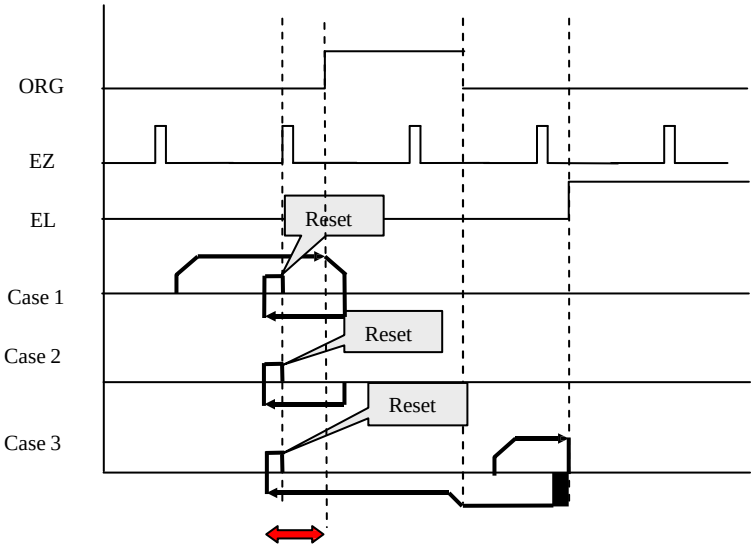


home_mode = 9 : Based on home mode 0

It can search the ORG edge signal at a specific direction automatically.



home_mode = 11 : Based on home mode 2



Abnormal stop when using these modes

During auto homing phases, there could be a stop command from limit switch, alarm switch or users' stop command. When it happened, the homing procedure will be terminated and return an error code in _8134_motion_done(). These error codes show the reason of termination. Notice that these codes will be cleared when users' program read it.

Normally, the _8134_motion_done() will show the phase codes during home searching. When it returns 0, it means homing is stopped. If there is no homing error codes returned on this duration, it means homing is successfully done. If there is an error code after stop, users must check the error code and the previous code from _8134_motion_done() to judge the reason of error and the phase of stop.

Please follow the sample program to read the return code correctly.

```
// Start home move
_8134_home_move(Axis);
while(1)
{
        // polling motion status
        Ret=_8134_motion_done(Axis)
        if( Ret >= 200 )  // Abnormal Stop during homing
        {
            // ALM_ON = 200
            // PEL_ON = 201
           // MEL_ON = 202
           // Stop command = 203
           // EMG command = 204
           // Unknow Stop =205
           // Other Stop = 206

           // This status will not be latched, it will be cleared after read
           break;
        }
        else if( Ret > 100 ) // Normal case : home phase returned
        {

        // Keep Homing Phase for debug when abnormal stop happen

                LastRet=Ret;

                // Auto homing Status ( phase will change during homing )
                // START = 100
                // END = 101
                // REVERSE = 102
                // FIND ORG = 103
                // ESCAPE ORG = 104
                // OFFSET ORG = 105
                // LEAVE ORG = 106
                // First MOVE = 107
```

```
            // Second MOVE =108
            // FIND EL = 109
            // FIND EZ = 110
        }
        else if( Ret == 0 )
        {
            // Normal End
            break;
        }

    } // end of while
```

Relative return codes of auto home return modes

return codes of _8134_motion_done(), defined in 8134err.h

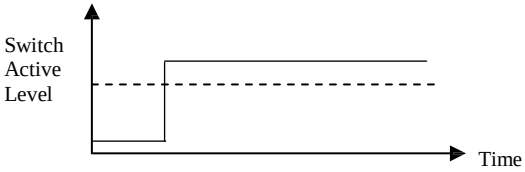
```
HOME_Error_ALM= 200
HOME_Error_PEL = 201
HOME_Error_MEL= 202
HOME_Error_SDSTOP= 203
HOME_Error_STOP = 204
HOME_Unknow_STOP = 205
HOME_Other_STOP = 206

HOME_START = 100
HOME_END = 101
HOME_REVERSE = 102
HOME_FIND_ORG = 103
HOME_ESCAPE_ORG = 104
HOME_OFFSET_ORG = 105
HOME_LEAVE_ORG = 106
HOME_ MOVE = 107
HOME_MOVE2 =108
HOME_FIND_EL = 109
HOME_FIND_EZ = 110
```

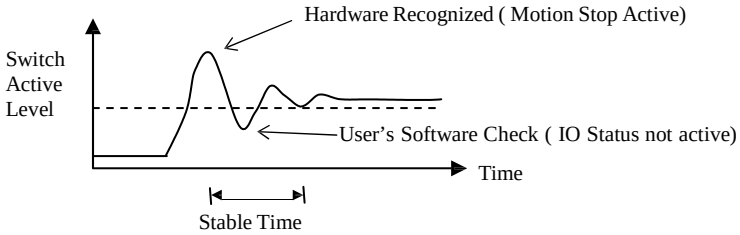
Bouncing Problem in I/O Switch

Sometimes, end-limit and ORG switches are mechanical type. Although there is a low pass filter in our input circuit, it could exist a bouncing problem if the bouncing frequency is very low. PCI-8134 provides a bouncing problem filter function, _8134_set_bounce_filter(AxisNo, Value). The default value is 10. It means this card will check 10 times during the switches changing their states. Every check is about 1ms~10ms depends on PC's performance. If users get any unknown error code(205) during homing, try to increase this value for optimization. In each phase of homing, PCI-8134 will place this checking after the phase is finished. There could be a motion pause situation during homing if the setting value is too high. That's normal situation.

Ideal Case:



Real Case:



Appendix B: 8134.DLL vs. 8134A.DLL

PCI-8134/PCI-8134A has two kinds of function library. They can't be mixed to use. If you are the first time or your project is a new created one, use the 8134a.DLL for development. The new library, 8134a.DLL, has many new functions and it also avoids function naming problem when co-working with other cards like PCI-8372/66. Of course, the old function library, 8134.DLL will continuously be maintenaned but will not support new functions in the future.

If users want to porting old codes to new function library. Please read the following comparation table for details.

B.1 Initialization

PCI8134.h (8134.lib)	***PCI8134a.h (8134a.lib)***
U16 W_8134_Initial(I32 card_number); U16 W_8134_InitialA(I16 *TotalCard);	I16 _8134_initial(I16 *existCards);(1)
U16 W_8134_Close(I32 card_number);	I16 _8134_close(void); (1)
U16 W_8134_Set_Config(U8 *fileName);	I16 _8134_config_from_file(U8 *fileName);
Void W_8134_Get_IRQ_Channel(U16 cardNo, U16 *irq_no);	I16 _8134_get_irq_channel(I16 CardNo, U16 *irq_no);
void W_8134_Get_Base_Addr(U16 cardNo, U16 *base_addr);	I16 _8134_get_base_addr(I16 CardNo, U16 *base_addr);
I16 version_info(I16 CardNo, U16 *HardwareInfo, I32 *SoftwareInfo, I32 *DriverInfo);	I16 _8134_version_info(I16 CardNo, U16 *HardwareInfo, I32 *SoftwareInfo, I32 *DriverInfo);

B.2 Pulse Input/Output Configuration

PCI8134.h (8134.lib)	*PCI8134a.h (8134a.lib)*
U16 set_pls_outmode(I32 axis, I32 pls_outmode);	I16 _8134_set_pls_outmode(I16 AxisNo, I16 pls_outmode);
U16 set_pls_iptmode(I32 axis, I32 pls_iptmode);	I16 _8134_set_pls_iptmode(I16 AxisNo, I16 pls_iptmode);
U16 set_cnt_src(I32 axis, I32 cnt_src);	I16 _8134_set_feedback_src(I16 AxisNo, I16 src);(3)

B.3 Continuously Motion Mode

PCI8134.h (8134.lib)	*PCI8134a.h (8134a.lib)*
U16 v_move(I16 axis, F64 str_vel, F64 max_vel, F64 accel);	I16 _8134_tv_move(I16 AxisNo, F64 StrVel, F64 MaxVel, F64 Tacc);
U16 sv_move(I16 axis, F64 str_vel, F64 max_vel, F64 Tlacc, F64 Tsacc);	I16 _8134_sv_move(I16 AxisNo, F64 StrVel, F64 MaxVel, F64 Tacc, F64 SVacc);
U16 v_change(I16 axis, F64 max_vel, F64 accel);	I16 _8134_v_change(I16 AxisNo, F64 Vel, F64 Time);
U16 v_stop(I16 axis, F64 decel);	I16 _8134_emg_stop(I16 AxisNo); I16 _8134_sd_stop(I16 AxisNo,F64 Tdec);
U16 set_sd_stop_mode(I16 axisno, I16 stop_mode);	I16 _8134_set_sd_stop_mode(I16 AxisNo, I16 sd_mode);
U16 fix_max_speed(I16 axis, F64 max_vel);	I16 _8134_fix_speed_range(I16 AxisNo, F64 MaxVel);(3)
U16 unfix_max_speed(I16 axis);	I16 _8134_unfix_speed_range(I16 AxisNo);(3)
I16 get_current_speed(I16 AxisNo, F64 *speed);	I16 _8134_get_current_speed(I16 AxisNo, F64 *speed);
F64 verify_speed(F64 StrVel,F64 MaxVel,F64 *minAccT,F64 *maxAccT, F64 MaxSpeed);	F64 _8134_verify_speed(F64 StrVel,F64 MaxVel,F64 *minAccT,F64 *maxAccT, F64 MaxSpeed);

B.4 Trapezoidal Motion Mode

PCI8134.h (8134.lib)	*PCI8134a.h (8134a.lib)*
U16 start_a_move(I16 axis, F64 pos, F64 str_vel, F64	I16 _8134_start_ta_move(I16 AxisNo, F64 Pos, F64 StrVel, F64

max_vel, F64 accel);	MaxVel, F64 Tacc, F64 Tdec);
U16 start_r_move(I16 axis, F64 distance, F64 str_vel, F64 max_vel, F64 accel);	I16 _8134_start_tr_move(I16 AxisNo, F64 Dist, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);
U16 start_t_move(I16 axis, F64 distance, F64 str_vel, F64 max_vel, F64 accel, F64 decel);	I16 _8134_start_tr_move(I16 AxisNo, F64 Dist, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);
U16 start_ta_move(I16 axis, F64 pos, F64 str_vel, F64 max_vel, F64 Tacc, F64 Tdec);	I16 _8134_start_ta_move(I16 AxisNo, F64 Pos, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);
U16 a_move(I16 axis, F64 pos, F64 str_vel, F64 max_vel, F64 accel);	Obsolete
U16 r_move(I16 axis, F64 distance, F64 str_vel, F64 max_vel, F64 accel);	Obsolete
U16 t_move(I16 axis, F64 distance F64 str_vel, F64 max_vel F64 accel, F64 decel);	Obsolete
U16 ta_move(I16 axis, F64 pos, F64 str_vel, F64 max_vel, F64 Tacc, F64 Tdec);	Obsolete
U16 wait_for_done(I16 axis);	Obsolete(use motion_done instead)
I16 set_rdp_mode(I16 AxisNO, I16 Mode);	I16 _8134_set_rdp_mode(I16 AxisNO, I16 Mode);

B.5 S-Curve Profile Motion

PCI8134.h (8134.lib)	***PCI8134a.h (8134a.lib)***
U16 start_s_move(I16 axis, F64 pos, F64 str_vel, F64 max_vel, F64 Tlacc, F64 Tsacc);	I16 _8134_start_sa_move(I16 AxisNo, F64 Pos, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
U16 s_move(I16 axis, F64 pos, F64 str_vel, F64 max_vel, F64 Tlacc, F64 Tsacc);	Obsolete
U16 start_rs_move(I16 axis, F64 distance, F64 str_vel, F64 max_vel, F64 Tlacc, F64	I16 _8134_start_sr_move(I16 AxisNo, F64 Dist, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec,

Tsacc);	F64 SVacc, F64 SVdec);
U16 rs_move(I16 axis, F64 distance, F64 str_vel, F64 max_vel, F64 Tlacc, F64 Tsacc);	Obsolete
U16 start_tas_move(I16 axis, F64 pos, F64 str_vel, F64 max_vel, F64 Tlacc, F64 Tsacc, F64 Tldec, F64 Tsdec);	I16 _8134_start_sa_move(I16 AxisNo, F64 Pos, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);(2)(4)
U16 tas_move(I16 axis, F64 pos, F64 str_vel, F64 max_vel, F64 Tlacc, F64 Tsacc, F64 Tldec, F64 Tsdec);	Obsolete

B.6 Multiple Axes Point to Point Motion

PCI8134.h (8134.lib)	*PCI8134a.h (8134a.lib)*
U16 start_move_all(I16 TotalAxes, I16 *map_array, F64 *pos, F64 *str_vel, F64 *max_vel, F64 *Tacc);	Obsolete(use set_ta_move_all and start_move_all instead)
U16 wait_for_all(I16 TotalAxes, I16 *map_array);	Obsolete
U16 move_all(I16 TotalAxes, I16 *map_array, F64 *pos, F64 *str_vel, F64 *max_vel, F64 *Tacc);	Obsolete
U16 start_sa_move_all(I16 len, I16 *map_array, F64 *pos, F64 *str_vel, F64 *max_vel, F64 *Tlacc, F64 *Tsacc);	Obsolete(use set_sa_move_all and start_move_all instead)

B.7 Linear Interpolated Motion

PCI8134.h (8134.lib)	*PCI8134a.h (8134a.lib)*
U16 move_xy(I32 cardNo, F64 x, F64 y);	Obsolete (use 2D function instead)
U16 move_zu(I32 cardNo, F64 z, F64 u);	Obsolete (use 2D function instead)
U16 start_move_xy(I32 cardNo,	I16 _8134_start_ta_move_xy(I16 CardNo, F64 PosX, F64 PosY,

F64 x, F64 y);	F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);
U16 start_move_zu(I32 cardNo, F64 z, F64 u);	I16 v_8134_start_ta_move_zu(I16 CardNo, F64 PosZ, F64 PosU, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);

B.8 Interpolaiton Parameters Configuring

PCI8134.h (8134.lib)	***PCI8134a.h (8134a.lib)***
U16 map_axes(I16 n_axes, I16 *map_array);	Obsolete
U16 set_move_speed(F64 str_vel, F64 max_vel);	Obsolete
U16 set_move_accel(F64 accel);	Obsolete
U16 set_move_ratio(I16 axis, F64 ratio);	I16 _8134_set_move_ratio(I16 AxisNo, F64 move_ratio);

B.9 Home Return Mode

PCI8134.h (8134.lib)	***PCI8134a.h (8134a.lib)***
U16 set_home_config(I32 axis,I32 home_mode,I32 org_logic,I32 org_latch,I32 EZ_logic);	I16 _8134_set_home_config(I16 AxisNo, I16 home_mode, I16 org_logic, I16 ez_logic, I16 ez_count, I16 erc_out);(1)
U16 home_move(I32 axis,F64 str_vel,F64 max_vel,F64 accel);	I16 _8134_home_move(I16 AxisNo, F64 StrVel, F64 MaxVel, F64 Tacc);
I16 set_org_offset(I16 AxisNo, F64 Offset);	I16 _8134_set_org_offset(I16 AxisNo, F64 Offset);

B.10 Manaul Pulser Motion

PCI8134.h (8134.lib)	***PCI8134a.h (8134a.lib)***
U16 set_manu_iptmode(I32 axis, I32 manu_iptmode, I32 op_mode);	I16 _8134_set_pulser_iptmode(I16 AxisNo, I16 InputMode, I16 Indep_Com);(1)(3)
U16 manu_move(I32 axis, F64 max_vel);	I16 _8134_pulser_vmove(I16 AxisNo,F64 SpeedLimit);(3)
U16 set_step_unit(I16 axisno, I16 unit);	I16 _8134_set_step_unit(I16 AxisNo, I16 UnitNo);

B.11 Motion Status

PCI8134.h (8134.lib)	*PCI8134a.h (8134a.lib)*
U16 motion_done(I16 axis);	I16 _8134_motion_done(I16 AxisNo);

B.12 Servo Drive Interface

PCI8134.h (8134.lib)	*PCI8134a.h (8134a.lib)*
U16 set_alm_logic(I32 axis, I32 alm_logic, I32 alm_mode);	I16 _8134_set_alm(I16 AxisNo, I16 alm_logic, I16 alm_mode);
U16 set_inp_logic(I32 axis, I32 inp_logic, I32 inp_enable);	I16 _8134_set_inp(I16 AxisNo, I16 inp_enable, I16 inp_logic);
U16 set_erc_enable(I32 axis, I32 erc_enable);	I16 _8134_set_erc_enable(I16 AxisNo, I16 erc_enable);
U16 set_sd_logic(I32 axis, I32 sd_logic, I32 sd_latch, I32 sd_enable);	I16 _8134_set_sd(I16 AxisNo, I16 enable, I16 sd_logic, I16 sd_latch, I16 sd_mode);(1)
U16 set_erc_enable(I32 axis, I32 erc_enable);	I16 set_erc_enable(I16 axis, I16 erc_enable);

B.13 I/O Control and Monitoring

PCI8134.h (8134.lib)	*PCI8134a.h (8134a.lib)*
U16 W_8134_Set_SVON(I32 axis, I32 on_off);	I16_8134_set_servo(I16 AxisNo, I16 on_off);
U16 get_io_status(I16 axis, U16 *io_sts);	I16 _8134_get_io_status(I16 AxisNo, U16 *io_sts);

B.14 Position Control

PCI8134.h (8134.lib)	*PCI8134a.h (8134a.lib)*
U16 get_position(I16 axis, F64 *pos);	I16 _8134_get_position(I16 AxisNo, F64 *pos);
U16 set_position(I16 axis, F64 pos);	I16 _8134_set_position(I16 AxisNo, F64 pos);
U16 get_command(I16 axis, F64 *pos);	I16 _8134_get_target_pos(I16 AxisNo, F64 *pos);(3)
U16 set_command(I16 axis, F64 pos);	I16 _8134_reset_target_pos(I16 AxisNo, F64 Pos);(3)

B.15 Interrupt Control

PCI8134.h (8134.lib)	*PCI8134a.h (8134a.lib)*
U16 W_8134_INT_Enable(I32 card_number);	I16 _8134_int_enable(I16 CardNo,HANDLE *phEvent);
U16 W_8134_INT_Disable(I32 card_number);	I16 _8134_int_enable(I16 CardNo);
Void W_8134_Set_INT_Control (U16 cardNo, U16 intFlag);	I16 _8134_int_control(I16 CardNo, I16 intFlag);
U16 set_int_factor(U16 axis, U32 int_factor);	I16 _8134_set_int_factor(I16 AxisNo, U32 int_factor);
U16 get_int_status(I32 axis, U32 *int_status);	I16 _8134_get_int_status(I16 AxisNo, U32 *int_factor);
I16 link_axis_interrupt(I16 AxisNo,void (__stdcall *callbackAddr)(void));	I16 _8134_link_axis_interrupt(I16 AxisNo,void (__stdcall *callbackAddr)(void));

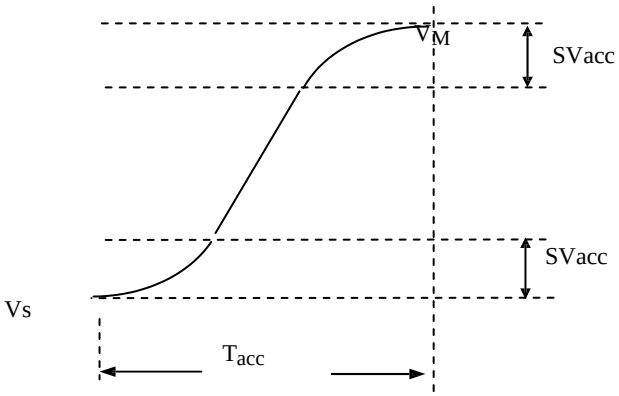
B.16 Soft-limit Checking

PCI8134.h (8134.lib)	*PCI8134a.h (8134a.lib)*
U16 set_sw_limit(I16 axisno,F64 p_limit, F64 n_limit);	I16 _8134_set_sw_limit(I16 axisno,F64 p_limit, F64 n_limit);
U16 unset_sw_limit(I16 axisno);	I16 _8134_unset_sw_limit(I16 axisno);

Note:

(1) The input arguments have been changed.

(2) The arguments in motion commands are different from last library. Users have to pay more attention to the mapping relations. The acceleration and deceleration arguments in s-curve motion command are changed a lot. The old ones use Tlacc, Tldec, Tsacc, and Tsdec as input arguments. However, the motion commands in new library adopt Tacc, SVacc, Tdec, and SVdec as input arguments.



(3) The library name has been changed entirely, but its function is the same with the old one.

(4) The motion commands in left side are simplified to the ones in the right side**.**

New Functions in PCI8134a.h (8134a.lib)

I16 _8134_set_org_latch(I16 AxisNo, I16 org_latch);
I16 _8134_start_move_all(I16 FirstAxisNo);
I16 _8134_stop_move_all(I16 FirstAxisNo);
I16 _8134_set_sync_option(I16 AxisNo, I16 sync_stop_on, I16

cstop_output_on);
I16 _8134_set_tr_move_all(I16 TotalAx, I16 *AxisArray, F64 *DistA, F64 *StrVelA, F64 *MaxVelA, F64 *TaccA, F64 *TdecA);
I16 _8134_set_ta_move_all(I16 TotalAx, I16 *AxisArray, F64 *PosA, F64 *StrVelA, F64 *MaxVelA, F64 *TaccA, F64 *TdecA);
I16 _8134_set_sr_move_all(I16 TotalAx, I16 *AxisArray, F64 *DistA, F64 *StrVelA, F64 *MaxVelA, F64 *TaccA, F64 *TdecA, F64 *SVaccA, F64 *SVdecA);
I16 _8134_set_sa_move_all(I16 TotalAx, I16 *AxisArray, F64 *PosA, F64 *StrVelA, F64 *MaxVelA, F64 *TaccA, F64 *TdecA, F64 *SVaccA, F64 *SVdecA);
I16 _8134_set_to_single_mode(I16 AxisX,I16 AxisY);
I16 _8134_set_org_logic(I16 AxisNo, I16 org_logic);
I16 _8134_set_feedback_error_detect(I16 AxisNo, I32 max_error);
I16 _8134_get_error_counter(I16 AxisNo, I16 *error_counter);
I16 _8134_reset_error_counter(I16 AxisNo);
I16 _8134_set_bounce_filter(I16 AxisNo, I16 Value);

Warranty Policy

Thank you for choosing ADLINK. To understand your rights and enjoy all the after-sales services we offer, please read the following carefully.

1. Before using ADLINK's products, please read the user manual and follow the instructions exactly. When sending in damaged products for repair, please attach an RMA application form.

2. All ADLINK products come with a two-year guarantee,free of repair charge.

   - The warranty period starts from the product's shipment date from ADLINK's factory
   - Peripherals and third-party products not manufactured by ADLINK will be covered by the original manufacturers' warranty
   - End users requiring maintenance services should contact their local dealers. Local warranty conditions will depend on the local dealers

3. Our repair service does not cover two-year guarantee while damages are caused by the following:

   a. Damage caused by not following instructions on user menus.
   b. Damage caused by carelessness on the users' part during product transportation.
   c. Damage caused by fire, earthquakes, floods, lightening, pollution and incorrect usage of voltage transformers.
   d. Damage caused by unsuitable storage environments with high temperatures, high humidity or volatile chemicals.
   e. Damage caused by leakage of battery fluid when changing batteries.
   f. Damages from improper repair by unauthorized technicians.
   g. Products with altered and damaged serial numbers are not entitled to our service.

h. Other categories not protected under our guarantees.

4. Customers are responsible for the fees regarding transportation of damaged products to our company or to the sales office.

5. To ensure the speed and quality of product repair, please download an RMA application form from our company website www.adlinktech.com. Damaged products with RMA forms attached receive priority.

For further questions, please contact our FAE staff.

ADLINK: service@adlinktech.com

Test & Measurement Product Segment: NuDAQ@adlinktech.com

Automation Product Segment: Automation@adlinktech.com

Computer & Communication Product Segment: NuPRO@adlinktech.com ; NuIPC@adlinktech.com