PCI-8164

Advanced 4 Axes Servo / Stepper Motion Control Card

User's Guide

Manual Rev. 1.00: February 19, 2001

Trademarks

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.adlink.com.tw http://www.adlinktechnology.com		
Service and technical support	service@adlink.com.tw		
TEL	+886-2-82265877	FAX	+886-2-82265717
Address	9F, No. 166, Jian Yi Road, Chungho City, Taipei, 235 Taiwan, R.O.C.		

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

How to Use This Guide

This manual is designed to help you use the PCI-8164. The manual describes how to modify various settings on the PCI-8164 card to meet your requirements. It is divided into six chapters:

Chapter 1	"Introduction", gives an overview of the product features, applications, and specifications.
Chapter 2,	"Installation", describes how to install the PCI-8164.
Chapter 3,	"Signal Connection", describes the connectors' pin assignment and how to connect the outside signal and devices with the PCI-8164.
Chapter 4,	"Operation Theorem ", describes detail operations of the PCI-8164.
Chapter 5,	"Motion Creator", describes how to utilize a Microsoft Windows based utility program to configure and test running the PCI-8164.
Chapter 6,	"C/C++ Function Library", describes high-level programming interface in C/C++ language. It helps programmer to control PCI-8164 in high level language style.
Chapter 7,	"Connection Example" shows some typical connection examples between PCI-8164 and servo driver and stepping driver.

1

Introduction

The PCI-8164 is an advanced 4 axes motion controller card with PCI interface. It can generate high frequency pulses (6.4MHz) to drive stepping/micro stepping motors and servo motors. In motion functions, it provides 2-axis circular, 4-axis linear interpolation, continuous interpolation with velocity continuity. Also, change position/speed on the fly are available in single axis operation. Multiple PCI-8164 cards can be used in one system. Incremental encoder interface on all four axes provide the ability to correct positioning errors generated by inaccurate mechanical transmissions, and with the help of on board FIFO, PCI-8164 can also perform precise and extremely fast position compare and trigger function without consuming CPU resource. 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-8164 card. PCI-8164 uses one ASICs (PCL6045) to perform 4 axes motion control. These ASICs are made of Nippon Pulse Motor incorporation. The motion control functions include linear and S-curve acceleration/deceleration, circular interpolation between two axes, linear interpolation between 2~4 axes, continuous motion, in positioning and 11 home return modes 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 software is equipped with the PCI-8164 card for supporting application development. The 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-8164 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.

Figure 1.2 is a flowchart that shows a recommending process of using this manual to develop an application. Please also refer the relative chapters for the detail of each step.

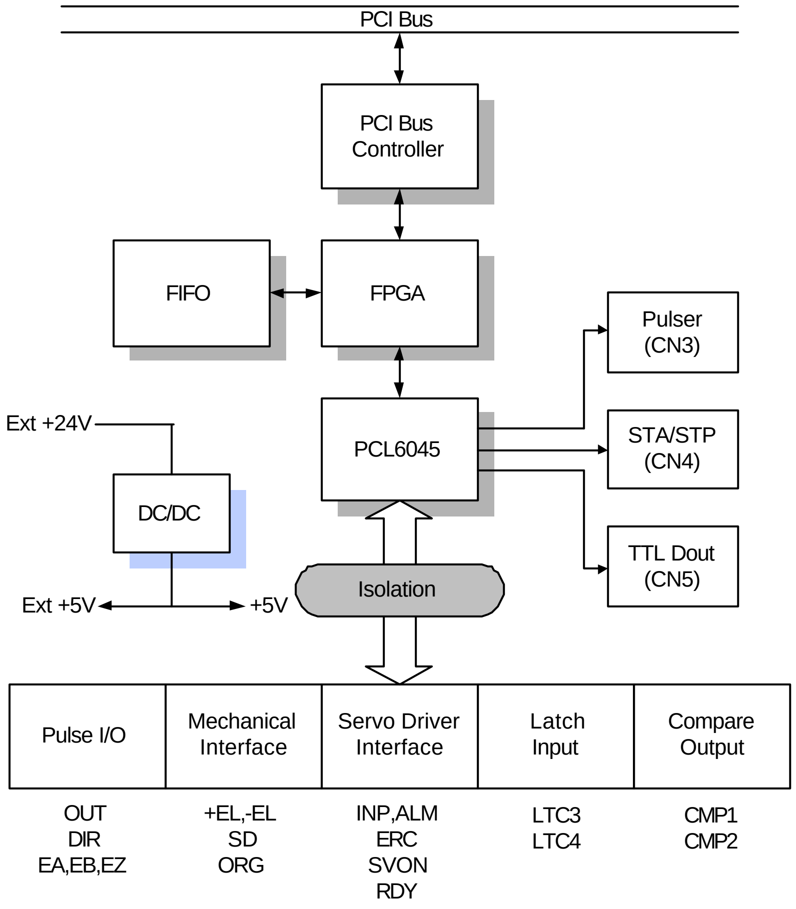


Figure 1.1 Block Diagram of PCI-8164

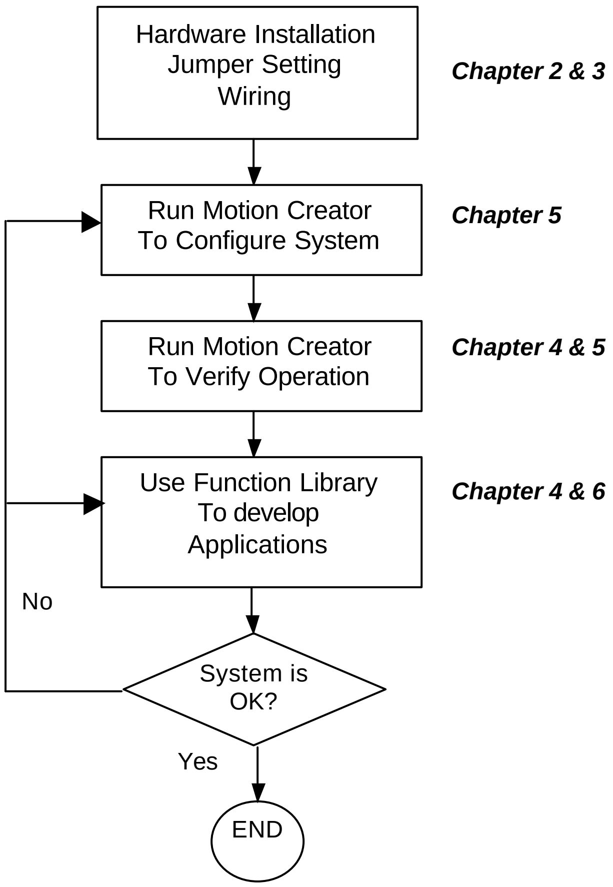


Figure 1.2 Flowchart of building an application

1.1 Features

The following lists summarize the main features of the PCI-8164 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 6.55 Mpps.
- Pulse output options: OUT/DIR, CW/CCW
- Programmable acceleration and deceleration time
- Trapezoidal and S-curve velocity profiles
- Any 2 of 4 axes circular interpolation.
- 2~4 axes linear interpolation.
- Continuous interpolation
- Change position and speed on the Fly.
- Change speed by compare trigger
- 13 home return modes
- Hardware backlash compensator and vibration suppression
- Software limit function
- 28-bit up/down counter for incremental encoder feedback.
- Home switch, index signal, positive and negative limit switches interface provided for all axes.
- 2 axes high speed position latch input
- 2 axes position compare trigger output with 4K FIFO auto-loading.
- All digital input and output signals are 2500Vrms isolated
- Programmable interrupt sources.
- Simultaneous start/stop motion on multiple axes.
- Manual pulser input interface.
- Software supports maximum up to 12 PCI-8164 cards (48 axes) operation.
- Compact, half size PCB.
- Motion Creator, Microsoft Windows based application development software.
- PCI-8164 Library and Utility for DOS library and Windows 95/98/NT/2000 DLL.

1.2 Specifications

- **Applicable Motors:**
  - Stepping motors.
  - AC or DC servomotors with pulse train input servo drivers.
- **Performance:**
  - Number of controllable axes: 4 axes.
  - Maximum pulse output frequency: 6.55Mpps, linear, trapezoidal or S-Curve velocity profile drive.
  - Internal reference clock: 19.66 MHz
  - Position pulse setting range: -134,217,728~ +134,217,728 pulses (28-bit).
  - Up / down counter counting range: 0~268,435,455 (28-bit.) or –134,217,728 to +134,217,727
  - Pulse rate setting range (Pulse Ratio = 1: 65535):

0.1 PPS to 6553.5 PPS.	(Multiplier = 0.1)
1 PPS to 65535 PPS.	(Multiplier = 1)
100 PPS to 6553500 PPS.	(Multiplier = 100)

- **I/O Signales:**
  - 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/PCS and ORG.
  - Servomotor interface I/O pins: INP, ALM and ERC.
  - Position latch input pin: LTC
  - Position compare output pin: CMP
  - 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-Purposed Output**
  - 6 TTL level Digital Output
- **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.
- Dimension: 185mm(L) X 98.4mm(H)

1.3 Software Supporting

1.3.1 Programming Library

For the customers who are writing their own programs, we provide MS-DOS Borland C/C++ programming library and Windows-95/98/ME/NT/2000 DLL for PCI-8164. These function libraries are shipped with the board.

1.3.2 Motion Creator

Refer to Chapter 5 for details.

2

Installation

This chapter describes how to install the PCI-8164. Please follow these steps below to install the PCI-8164.

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

2.1 What You Have

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

- PCI-8164: advanced 4 Axes Servo / Stepper Motion Control Card
- ADLINK All-in-one Compact Disc
- +24V power input cable (for CN1) accessory.

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-8164 Outline Drawing

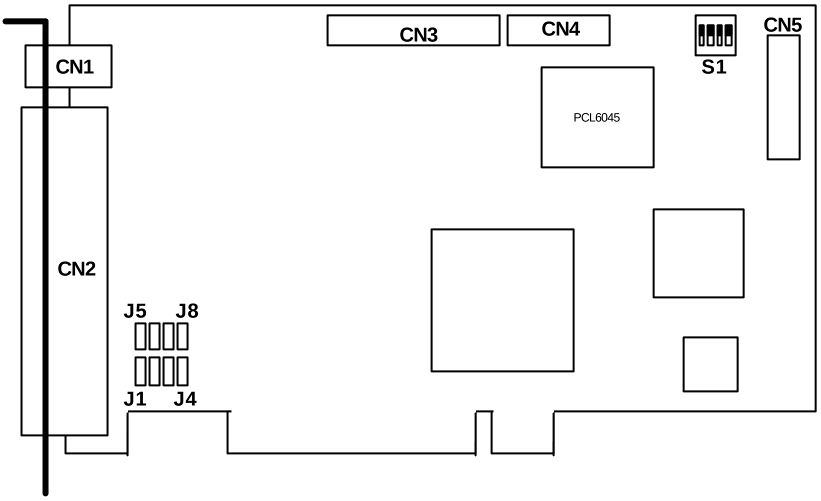
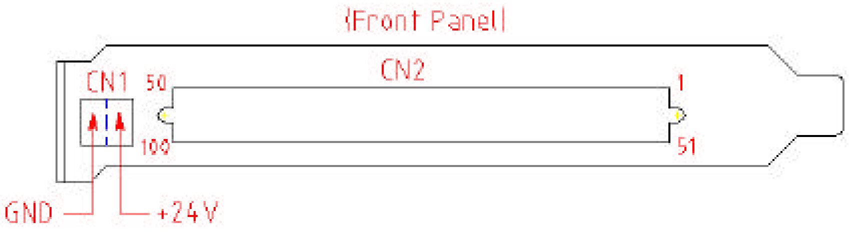


Figure 2.1 PCB Layout of the PCI-8164

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



2.3 Hardware Installation

2.3.1 Hardware configuration

PCI-8164 has 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-8164 can be used in any PCI slot.

2.3.3 Installation Procedures

1. Read through this manual, and setup the jumper according to your application
2. Turn off your computer, Turn off all accessories (printer, modem, monitor, etc.) connected to computer.

Remove the cover from your computer.

3. Select a 32-bit PCI expansion slot. PCI slots are shorter than ISA or EISA slots and are usually white or ivory.
4. Before handling the PCI-8164, discharge any static buildup on your body by touching the metal case of the computer. Hold the edge and do not touch the components.
5. Position the board into the PCI slot you selected.
6. Secure the card in place at the rear panel of the system unit using screw removed from the slot.

2.3.4 Trouble shooting:

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 come with your system.

2.4 Software Driver Installation

Step 1: Auto-Run from ADLINK All-In-One CD, choose NuDAQ-PCI, then PCI-8164

Step 2: Follow the procedures of installer.

Step 3: After setup completion, restart windows.

2.5 CN1 Pin Assignments: External Power Input

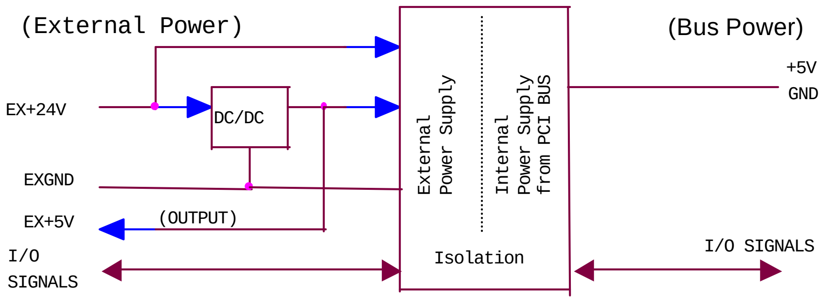
CN1 Pin No	Name	Description
1	EXGND	Grounds of the external power.
2	EX+24V	External power supply of +24V DC ± 5%

Notes:

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:0.08mm^2 to 0.32mm^2 (AWG28 to AWG22)
Naked wire length:10mm standard

The following diagram shows the external power supply system of the PCI-8164. The external +24V power must be provided, an on-board regulator generates +5V for both internal and external usage.



2.6 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	VPP	O	+5V power supply output	51	VPP	O	+5V power supply output
2	GND		Ext. power ground	52	GND		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	GND		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	VPP	O	+5V power supply output	69	VPP	O	+5V power supply output
20	GND		Ext. power ground	70	GND		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	GND		Ext. power ground	80	GND		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	CMP1	O	Position compare output ①	89	LTC3	I	Position latch input ③
40	SD/PCS1	I	Ramp-down signal ①	90	SD/PCS3	I	Ramp-down signal ③
41	ORG1	I	Origin signal, ①	91	ORG3	I	Origin signal, ③
42	GND		Ext. power ground	92	GND		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	CMP2	O	Position compare output ②	95	LTC4	I	Position latch input, ④
46	SD/PCS2	I	Ramp-down signal ②	96	SD/PCS4	I	Ramp-down signal ④
47	ORG2	I	Origin signal, ②	97	ORG4	I	Origin signal, ④
48	GND		Ext. power ground	98	GND		Ext. power ground
49	GND		Ext. power ground	99	E_24V	0	Ext. power supply, +24V
50	GND		Ext. power ground	100	E_24V	0	Ext. power supply, +24V

2.7 CN3 Pin Assignments: Manual Pulser Input

The signals on CN3 are 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.

2.8 CN4 Pin Assignments: Simultaneous Start/Stop

The signals on CN3 are for sim ultaneously 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.

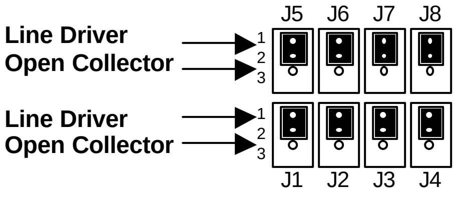
2.9 CN5 Pin Assignment : TTL Output

The signals on CN5 are for general-purposed TTL output signals.

Pin No.	Name	Function
1	DGND	Digital ground
2	DGND	Digital ground
3	ED0	Digital Output 0
4	ED1	Digital Output 1
5	ED2	Digital Output 2
6	ED3	Digital Output 3
7	ED4	Digital Output 4
8	ED5	Digital Output 5
9	VCC	VCC +5V
10	N.C.	No use

2.10 Jumper Setting for Pulse Output

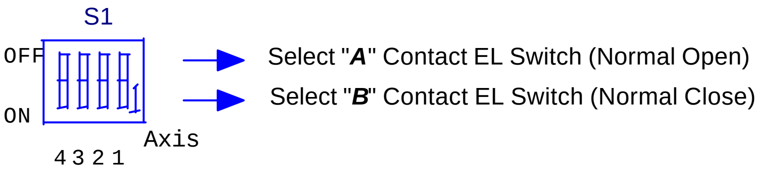
The J1~J8 is used to set the signal type of the pulse output signals (DIR and OUT). The output signal type could be differential line driver output or open collector output. Please refer to section 3.1 for details of the jumper setting. The default setting is the differential line driver mode.



2.11 Switch Setting for EL Logic

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." "

Placement of S1 Switch on Board


S1
OFF
ON
4 3 2 1
Axis
Select "***A***" Contact EL Switch (Normal Open)
Select "***B***" Contact EL Switch (Normal Close)

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-8164 and the motor drivers.

This chapter contains the following sections:

3.1 Pulse Output Signals OUT and DIR

There are 4 axes pulse output signals on PCI-8164. 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.

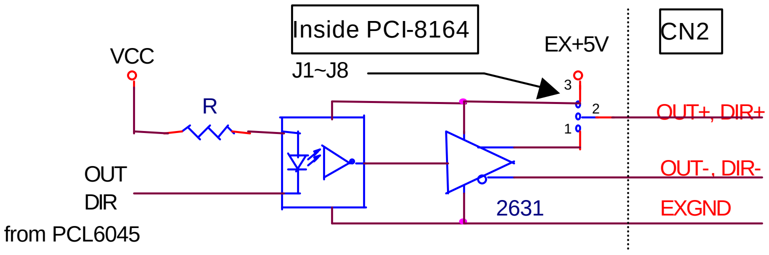
CN2 Pin No.	Signal Name	Description	Axis #
3	**OUT1+**	Pulse signals (+)	①
4	**OUT1-**	Pulse signals (-)	①
5	**DIR1+**	Direction signal(+)	①
6	**DIR1-**	Direction signal(-)	①
21	**OUT2+**	Pulse signals (+)	②
22	**OUT2-**	Pulse signals (-)	②
23	**DIR2+**	Direction signal(+)	②
24	**DIR2-**	Direction signal(-)	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.

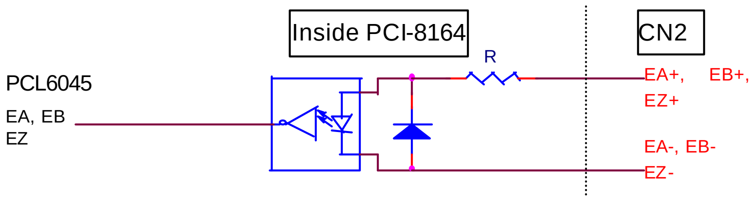
3.2 Encoder Feedback Signals EA, EB and EZ

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, and EZ signals are shown as follows.

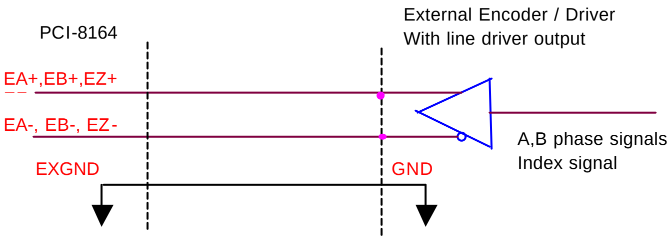


Please note that the voltage across every differential pair of encoder input signals (EA+, EA-), (EB+, EB-) and (EZ+, EZ-) should be at least 3.5V or higher. Therefore, you have to take care of the driving capability when connecting with the encoder feedback or motor driver feedback. The differential signal pairs will be converted to digital signal EA, EB and EZ to connect to PCL6045 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-8164 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.

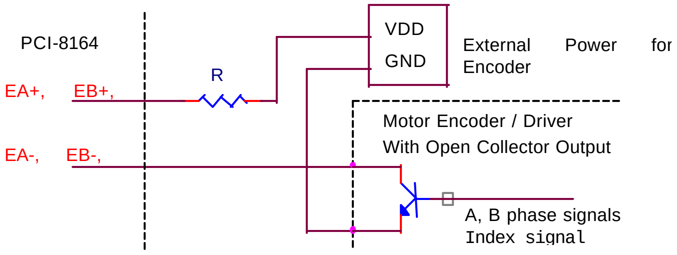


◆ Connection to Open Collector Output

To connect with open collector output, an external power supply is necessary. Some motor drivers also provide the power source. The connection between PCI-8164, 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-8164 input circuit. The following table lists the suggested resistor value according to the encoder power supply.

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

If=6mA max.



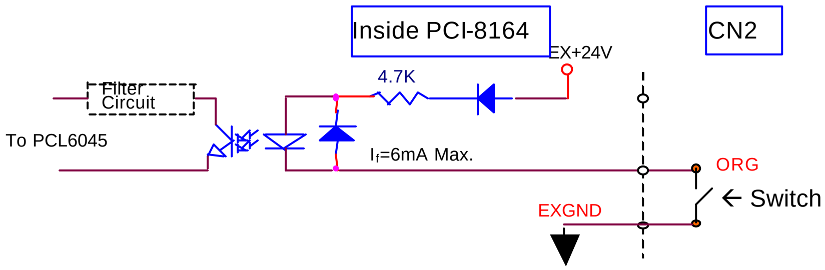
For more detail operation of the encoder feedback signals, please refer to section 4.4.

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.



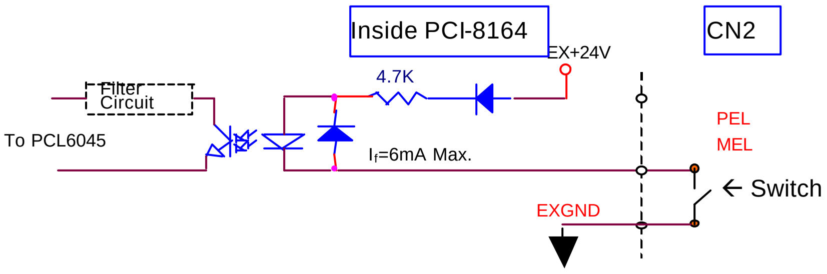
When the motion controller is operated at the home return mode, the ORG signal is used to stop the control output signals (OUT and DIR). For the detail operation of the ORG, please refer to section 4.3.3.

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-8164 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.

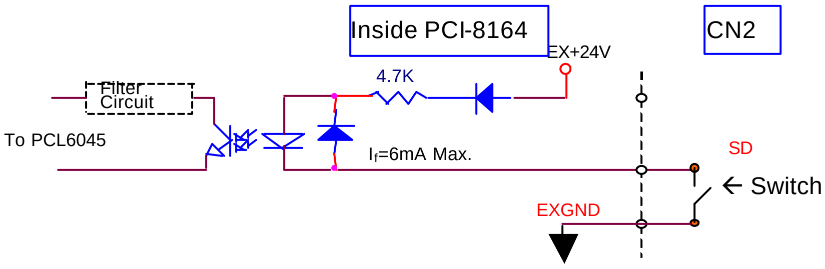


3.5 Ramping-down & PCS

There is a SD/PCS signal, for every of the 4 axis. The relative signal name, pin number and axis number are shown in the following table.

CN2 Pin No	Signal Name	Axis #
40	SD1/PCS1	①
46	SD2/PCS2	②
90	SD3/PCS3	③
96	SD4/PCS4	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/PCS operation, please refer to section 4.3.1.

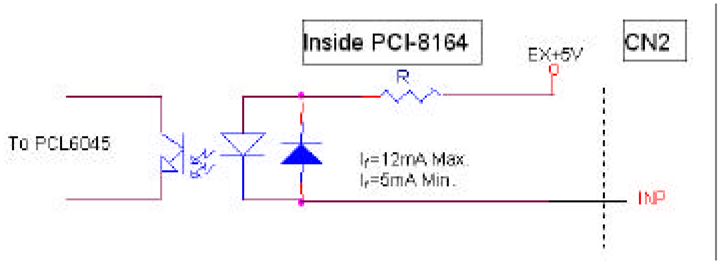


3.6 In-position Signal INP

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 is shown in the following diagram.



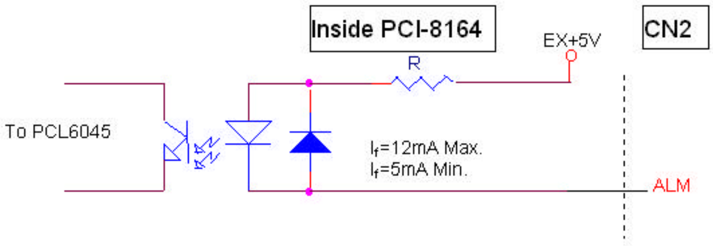
The in-position signals are usually from servomotor drivers, which usually provide open collector output signals. The external circuit must provide at least 5 mA current sink capability to drive the INP signal active. For more details of the INP signal operating, please refer to section 4.2.1.

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

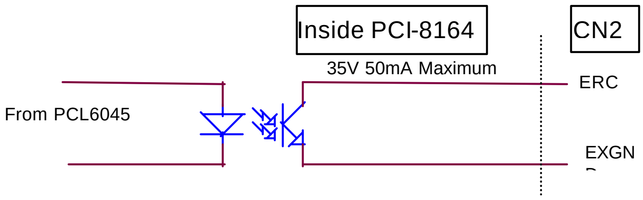
The deviation counter clear signal (ERC) is active in the following 4 situations:

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.

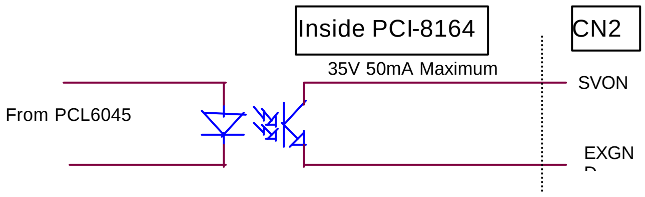


3.9 General-purpose Signal SVON

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.

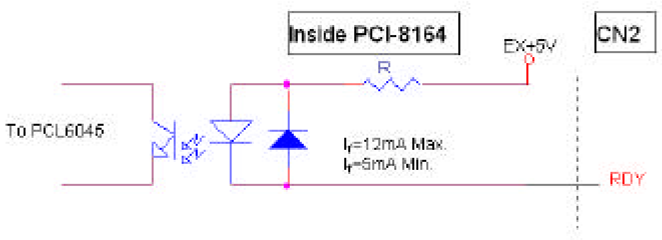


3.10 General-purpose Signal RDY

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
79	RDY4	④

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



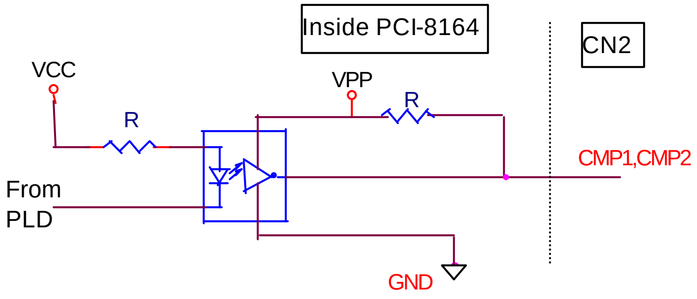
3.11 Position compare output pin: CMP

The PCI-8164 provides 2 compare output channels, CMP, and they are for the first 2 axes, ① & ②, only. The compare output will generate a pulse signal when encoder counter reached the value pre-set by user.

The CMP channel is in CN2. The relative signal name, pin number and axis number are shown in the following table.

CN2 Pin No	Signal Name	Axis #
39	CMP1	①
45	CMP2	②

The following wiring diagram is for the CMP of the first 2 axes.



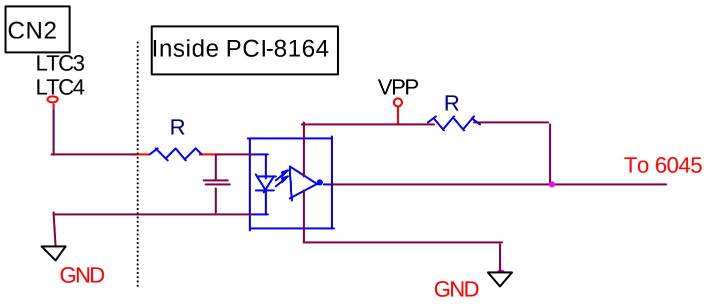
3.12 Position latch input pin: LTC

The PCI-8164 provides 2 position latch input channels, LTC, and they are for the last 2 axes, ③ & ④, only. The LTC signal will trigger the counter-value-capturing functions, which gives a precise position determination.

The CMP channel is in CN2. The relative signal name, pin number and axis number are shown in the following table.

CN2 Pin No	Signal Name	Axis #
89	LTC3	③
95	LTC4	④

The following wiring diagram is for the LTC of the last 2 axes.

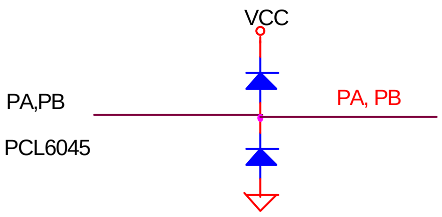


3.13 Pulser Input Signals PA and PB

The PCI-8164 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 PCL6045. The interface circuits are shown as follows.

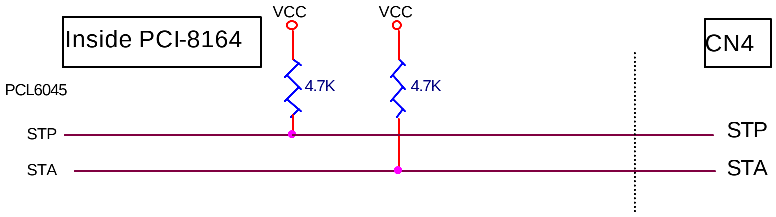


If the signal voltage of pulser is not +5V or if the pulser is distantly placed, it is recommended to put a photo coupler or line driver in between. Also, +5V and GND power lines of CN3 are direct from the PCI bus. Please carefully use these signals because they are not isolated.

3.14 Simultaneously Start/Stop Signals STA and STP

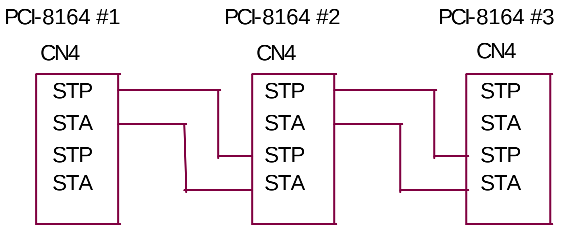
The PCI-8164 provides the STA and STP signals, which enable simultaneous start/stop of motions on multiple axes. The STA and STP signals are on the CN4.

The following diagram shows the on-board circuits. The STA and STP signals of the four axes are tight together respectively.

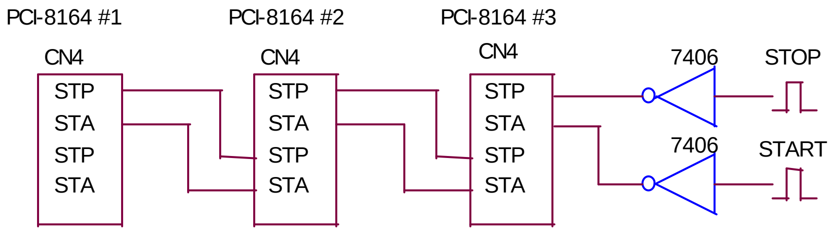


The STP and STA signals are both input and output signal. To operate the simultaneously start and stop action, both software control and external control are possible. By the software control, the signals can be generated from any one of the PCL6045, and other chip will start and stop simultaneously if proper programmed. You can also use an external open collector or switch to drive the STA/STP signals for simultaneous start/stop.

If there are two or more PCI-8164 cards, cascade CN4 connectors of all cards for simultaneous start/stop control on all concerned axes is possible. In this case, connect CN4 as follows.



To let an external signal to initiate simultaneous start/stop, connect the 7406 (open collector) or the equivalent circuit as follows.



3.15 Daughter Board Connector

The CN2 connector of PCI-8164 can be connected with DIN-100S, including a cable ACL-102100 (a 100-pin SCSI-II cable). DIN-100S is a general purpose DIN-socket with 100-pin SCSI-II connector. It has easily wiring screw terminal and easily installation DIN socket that can be mounted on DIN-rails

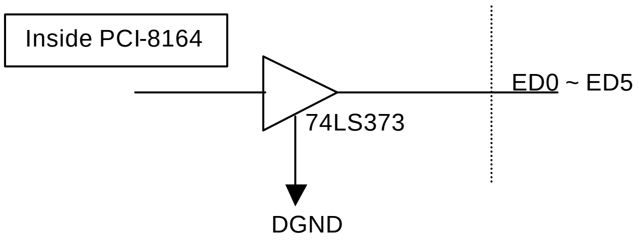
Please check the NuDAQ catalog by ADLINK for further information of DIN-100S

3.16 General-purposed TTL Output

The PCI-8164 provides 6 general-purposed TTL digital output. The TTL output is in CN5. The relative signal name, pin number and axis number are shown in the following table.

Pin No.	Name	Function
1	DGND	Digital ground
2	DGND	Digital ground
3	ED0	Digital Output 0
4	ED1	Digital Output 1
5	ED2	Digital Output 2
6	ED3	Digital Output 3
7	ED4	Digital Output 4
8	ED5	Digital Output 5
9	VCC	VCC +5V

The following wiring diagram is for the LTC of the last 2 axes.



4

Operation Theorem

This chapter describes the detail operation of the PCI-8164 card. Contents of the following sections are as following.

4.1 Motion Control Modes

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

- 4.1.1 Pulse Command Output
- 4.1.2 Velocity mode motion for one axis
- 4.1.3 Trapezoidal motion for one axis
- 4.1.4 S-Curve profile motion for one axis
- 4.1.5 Linear interpolation for 2~4 axes
- 4.1.6 Circular interpolation for 2 axes
- 4.1.7 Continuous interpolation for 2 axes
- 4.1.8 Home return mode for one axis
- 4.1.9 Manual pulser mode for one axis

4.1.1 Pulse Command Output

The PCI-8164 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: **_8164_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:

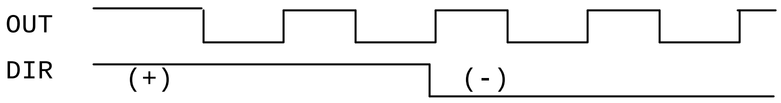
Mode	Output of OUT pin	Output of DIR pin
Dual pulse output (CW/CCW)	Pulse signal in plus (or CW) direction	Pulse signal in minus (or CCW) direction
Single pulse output (OUT/DIR)	Pulse signal	Direction signal (level)

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.

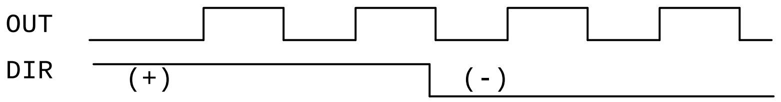
Single Pulse Output Mode(OUT/DIR Mode)

In this mode, the OUT signal is for the command pulse (position or velocity) chain. The numbers of OUT pulse represent the 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 diagrams show the output waveform. It is possible to set the polarity of pulse chain.

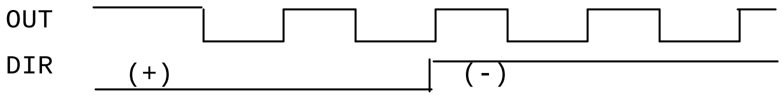
pls_outmode = 0:



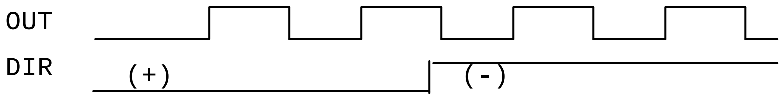
pls_outmode = 1:



pls_outmode = 2:



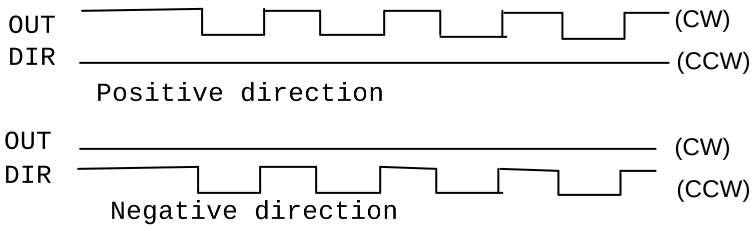
pls_outmode = 3:



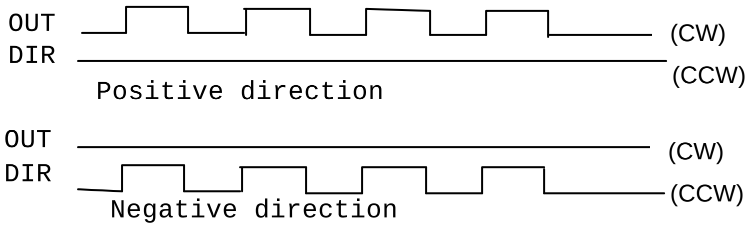
Dual Pulse Output Mode(CW/CCW Mode)

In this mode, the waveform of the OUT and DIR pins represent 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.

pls_outmode = 4:



pls_outmode = 5:



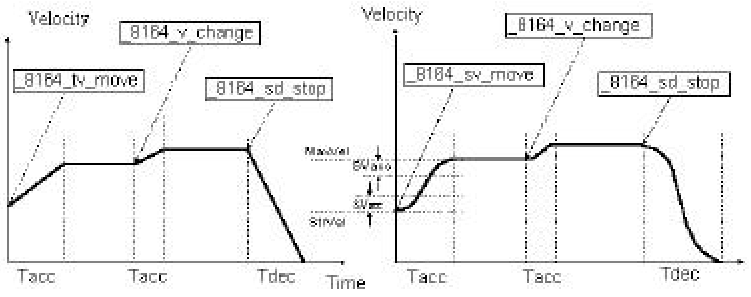
Relative Function:

_8164_set_pls_optmode(): Refer to section 6.4

4.1.2 Velocity mode motion

This mode is used to operate one axis motor at Velocity mode motion. The output pulse accelerates from a starting velocity (StrVel) to the specified constant velocity (MaxVel). The **_8164_tv_move()** function is used to accelerate constantly while the **, _8164_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 **_8164_v_change()** is used to change speed during moving. Before this function is applied, be sure to call ***_8164*_fix_speed_range()**. Please refer to section 4.6 for more detail explanation. The **_8164_sd_stop()** is used to decelerate the motion to stop. The **_8164_emg_stop()** function is used to immediately stop the motion. Those change or stop functions follow the same velocity profile as its original move functions, tv_move or sv_move. The velocity profile is shown as following.

Note: The v_change and stop functions can also be applied to **Preset Mode** (both trapezoidal, refer to 4.1.3 and S-curve Motion, refer to 4.1.4) or **Home Mode** (refer to 4.1.8)**.**



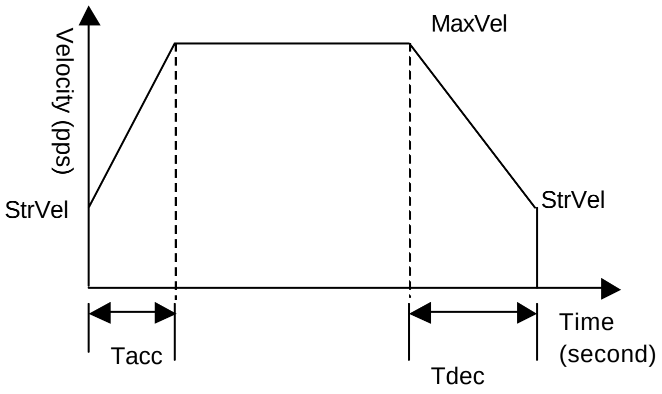
Relative Functions:

_8164_tv_move(),_8164_sv_move(),_8164_v_change(),_8164_sd_stop(), _8164_emg_stop(),_8164_fix_speed_range(),_8164_unfix_speed_range() : 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 function _8164_motion_done()is used to check whether the movement is complete.

The following diagram shows the trapezoidal profile.



There are 2 trapezoidal point-to-point functions supported by PCI-8164. In the **_8164_start_ta_move()** function, 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, the "External encoder feedback (EA, EB pins)" should be set in **_8164_set_feedback_src()** function. And the ratio between command pulses and external feedback pulse input must be appropriately set by **_8164_set_move_ratio()** function.

In the **_8164_start_tr_move()** function, the relative displacement must be given in the unit of pulse. Unsymmetrical trapezoidal velocity profile (Tacc is not equal Tdec) can be specified in both **_8164_start_ta_move()** and **_8164_start_tr_move()** functions.

The StrVel and MaxVel 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 pulse" to calculate the physical value of the relative velocity or acceleration parameters. The following formula gives the basic relationship between these parameters.

MaxVel = StrVel + accel*Tacc;
StrVel = MaxVel + decel *Tdec;

where accel/decel represents the acceleration/deceleration rate in unit of pps/sec^2. 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 servomotor with a AB phase encoder is used in 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-8164?

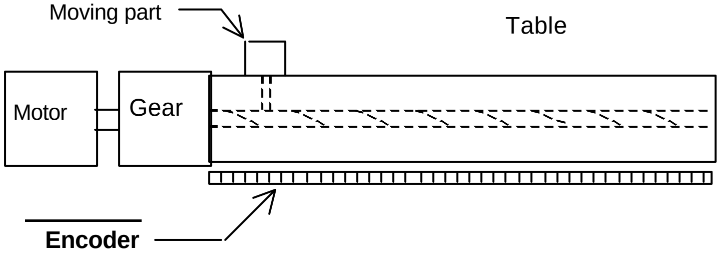
Answer:

MaxVel = 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. If 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:

_8164_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-8164 can't recognize the actual absolute position during motion.

Relative Functions:

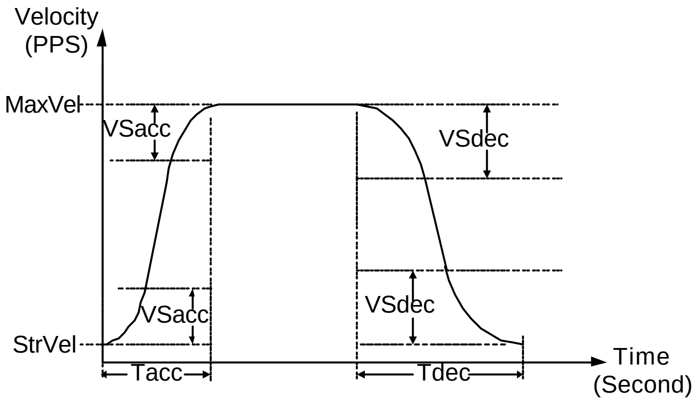
_8164_start_ta_move(),_8164_start_tr_move() : Refer to section 6.6
_8164_motion_done(): Refer to section 6.11
_8164_set_feedback_src(): Refer to section 6.4
_8164_set_move_ratio(): Refer to section 6.6

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, in unit of pulse.
Dist :	moving distance in relative mode, in unit of pulse.
StrVel:	specify the start velocity, in unit of PPS.
MaxVel:	specify the maximum velocity, in unit of PPS.
	Tacc pecify the time for acceleration (StrVel → MaxVel), in unit of second.
Tdec:	specify the time for deceleration (MaxVel → StrVel), in unit of second.
SVacc :	specify the S-curve region during acceleration, in unit of PPS.
SVdec :	specify the S-curve region during deceleration, in unit of PPS.



Normally, the accel/decel period consist of 3 regions, two Vsacc/VSdec and one linear. During the VSacc/VSdec, the jerk (second derivative of velocity) is constant, and, during the linear region, the acceleration (first derivative of velocity) is constant. In the first constant jerk region during acceleration, the velocity goes from StrVel to (StrVel + Svacc). In the second constant jerk region during acceleration, the velocity goes from (MaxVel – StrVel) to MaxVel. Between them, the linear region accelerates velocity from (StrVel + VSacc) to (MaxVel - VSacc) constantly. The deceleration period gets similar rule.

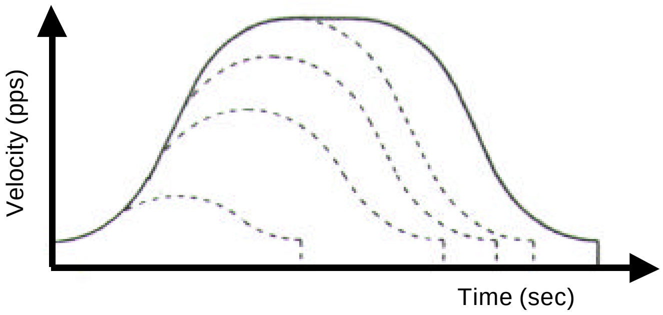
Special case:

if user wants to vanish the linear region, the VSacc/VSdec must be assigned "0" rather than 0.5*(MaxVel-StrVel).

Remember that the VSacc/VSdec is in unit of PPS and it should always keep in the range of [0 ~ (MaxVel - Strvel)/2], where "0" means no linear region.

The S-curve profile motion functions are designed to always produce smooth motion. If the time for 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 MaxVel), the maximum velocity is automatically lowered (see the following Figure).

The rule is to lower the value of MaxVel and the Tacc, Tdec, VSacc, VSdec automatically, and keep StrVel, acceleration and jerk unchanged. It is also applicable to Trapezoidal profile motion.



Relative Functions:

_8164_start_sr_move(),_8164_start_sa_move() : Refer to section 6.6
_8164_motion_done(): Refer to section 6.11
_8164_set_feedback_src(): Refer to section 6.4
_8164_set_move_ratio(): Refer to section 6.6

The Following table shows the difference between all the single axis motion functions, including **Preset Mode** (both trapezoidal and S-curve Motion) and **constant velocity mode**.

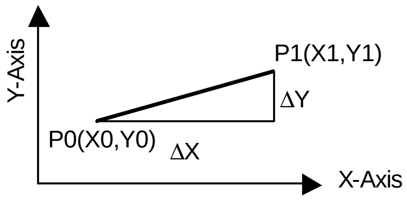
	Velocity Profile		Relative	Absolute
	Trapezoidal	S-Curve		
*_8163_**tv**_move*	∨		-----------	-----------
*_8163_**sv**_move*		∨	-----------	-----------
_8164_v_change	∨	∨	-----------	-----------
_8164_sd_stop	∨	∨	-----------	-----------
_8164_emg_stop()	-----------	------------	-----------	-----------
_8164_start_**ta**_move	∨			∨
_8164_start_**tr**_move	∨		∨	
_8164_start_**sr**_move		∨	∨	
_8164_start_**sa**_move		∨		∨

4.1.5 Linear interpolation for 2~4 axes

In this mode, any 2 of the 4, 3 of the 4 or all the 4 axes may be chosen to perform linear interpolation. " Interpolation between multi-axes" means these axes " start simultaneously, and reach their ending points at the same time" . Linear means the ratio of speed of every axis is a constant value.

2 axes linear interpolation

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



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

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

When calling the 2 axes linear interpolation functions, it is the **vector speed** to define the start velocity, **StrVel**, and maximum velocity, **MaxVel**, Both trapezoidal and S-curve profile are available.

Example:

_8164_start_tr_move_xy(0 , 30000.0 , 40000.0 , **1000.0 , 5000.0** , 0.1,0.2)

It will cause the X,Y axes (axes 0 & 1) of Card 0 to perform a linear interpolation movement, in which:

ΔX = 30000 pulse
ΔY = 40000 pulse
Start vector speed=1000pps, X speed=600pps, Y speed = 800 pps
Max. vector speed =5000pps, X speed=3000pps,Y speed = 4000pps
Acceleration time = 0.1 sec
Deceleration time = 0.2 sec

There are two groups of functions that provide 2 axes linear interpolation. The first group divides the 4 axes into XY (axis 0 & axis 1) and ZU(axis 2 & axis 3). By calling these functions, the target axes are already assigned.

_8164_start_tr_move_xy(), _8164_start_tr_move_zu(),
_8164_start_ta_move_xy(), _8164_start_ta_move_zu(),
_8164_start_sr_move_xy(), _8164_start_sr_move_zu(),
_8164_start_sa_move_xy(), _8164_start_sa_move_zu(),
: Refer to section 6.7

The second group allows user to freely assign the 2 target axes.

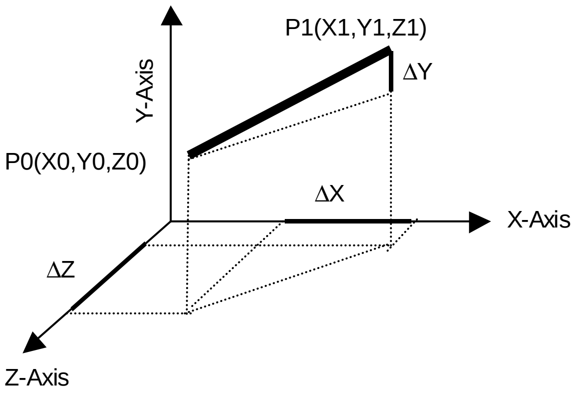
_8164_start_tr_line2(), _8164_start_sr_line2(),
_8164_start_ta_line2(),_8164_start_sa_line2(),
: Refer to section 6.7

The characters "t", "s", "r", "a" after **_8164_start** means:

t – Trapezoidal profile
s – S-Curve profile
r – Relative motion
a – Absolute motion

3 axes linear interpolation

Any 3 of the 4 axes of PCI-8164 may perform 3 axes linear interpolation. As the figure below, 3 axes linear interpolation means to move the XYZ (if axes 0, 1, 2 are selected and assigned to be X, Y, Z respectively) position from P0 to P1 and start and stop simultaneously. The path is a straight line in space.



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

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

When calling those 3 axes linear interpolation functions, it is the vector speed to define the start velocity, **StrVel**, and maximum velocity, **MaxVel**. Both trapezoidal and S-curve profile are available.

For example:

_8164_start_tr_line3(….,1000.0 /*ΔX */ , 2000.0/*ΔY */, 3000.0 /*DistZ*/, 100.0 /*StrVel*/, 5000.0 /* MaxVel*/, 0.1/*sec*/, 0.2 /*sec*/)

ΔX = 1000 pulse
ΔY = 2000 pulse
ΔZ = 3000 pulse
Start vector speed=100pps, X spped= 100/$\sqrt{14}$ = 26.7 pps
Y spped = 2*100/$\sqrt{14}$ = 53.3 pps
z spped = 3*100/$\sqrt{14}$ = 80.1 pps
Max. vector speed =5000pps,X spped= 5000/$\sqrt{14}$ = 1336 pps
Y spped = 2*5000/$\sqrt{14}$ = 2672 pps
z spped = 3*5000/$\sqrt{14}$ = 4008 pps

These functions related to 3 axes linear interpolation are listed below:

_8164_start_tr_line3(), _8164_start_sr_line3()
_8164_start_ta_line3() , _8164_start_sa_line3()
: Refer to section 6.7

The characters " t" , " s" , " r" , " a" after **_8164_start** means:

t – Trapezoidal profile
s – S-Curve profile
r – Relative motion
a – Absolute motion

4 axes linear interpolation

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

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

The functions related to 4 axes linear interpolation are listed below:

_8164_start_tr_line4(), _8164_start_sr_line4()
_8164_start_ta_line4(),_8164_start_sa_line4()
: Refer to section 6.7

The characters " t" , " s" , " r" , " a" after **_8164_start** means:

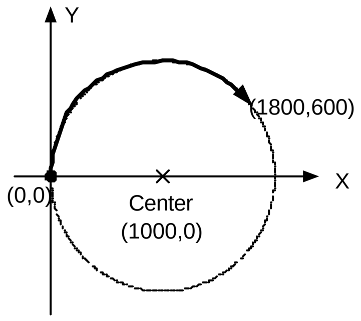
t – Trapezoidal profile
s – S-Curve profile
r – Relative motion
a – Absolute motion

4.1.6 Circular interpolation for 2 axes

Any 2 of the 4 axes of PCI-8164 can perform circular interpolation. As the example below, the circular interpolation means XY (if axes 0, 1 are selected and assigned to be X, Y respectively) axes simultaneously start from initial point, (0,0) and stop at end point,(1800,600). The path between them is an arc, and the MaxVel is the tangent speed.

Example:

_8164_start_a_arc_xy(0 /*card No*/, **1000,0 /***center X*/**, 0 /***center Y*/, **1800.0** /* End X */, **600.0 /***End Y */ ,1000.0 /* MaxVel */)



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

Center point: The coordinate of the center of arc (In absolute mode) or

The off_set distance to the center of arc(In relative mode)

End point: The coordinate of end point of arc (In absolute mode) or

The off_set distance to center of arc (In relative mode)

Direction: The moving direction, either CW or CCW.

It is not necessary to set radius or angle of arc, since the information above gives enough constrains. The arc motion stopped when either of the 2 axes reached end point.

There are two groups of functions that provide 2 axes circular interpolation. The first group divides the 4 axes into XY (axis 0 & axis 1) and ZU(axis 2 & axis 3). By calling these functions, the target axes are already assigned.

_8164_start_r_arc_xy(), _8164_start_r_ arc _zu(),
_8164_start_a_ arc _xy(), _8164_start_a_ arc _zu(),
: Refer to section 6.8

The second group allows user to freely assign any 2 target axes.

_8164_start_r_arc2(),_8164_start_a_arc2(),
: Refer to section 6.8

4.1.7 Continuous motion

The PCI-8164 allow user to perform continuous motion. Both single axis movement (section 4.1.3: Trapezoidal, section 4.1.4: S-Curve) and multi-axis interpolation (4.1.5: linear interpolation, 4.1.6: circular interpolation) can be extended to be continuous motion.

For example, if user calls the follow function to perform a single axis preset motion:

_8164_start_ta_move(0,50000.0,100.0,30000.0,0.1,0.0)

It will cause the axis "0" to move to position "50000.0", before the axis arrives, user can call a second pressed motion:

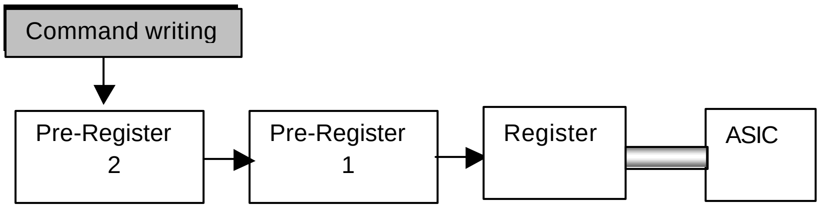
_8164_start_tr_move(0,20000.0,100.0,30000.0,0.0,0.2)

The second function call won't affect the first one, actually it will be executed and write into the pre-register in PCI-8164. After the first move is finished, PCI-8164 will continue the second move according to the pre-register value. So, no time interval exists between these two moves. And pulses will be continuously generated at the instant of position "50000.0"

The working theory of continuous motion is described below:

working theory of continuous motion

The following diagram shows the register data flow of PCI-8164.



Step 0: All Register and Pre-Register is empty.

Step 1: The first motion is executed and CPU writes corresponding values into pre-register 2.

_8164_start_ta_move(0,50000.0,100.0,30000.0,0.1,0.0)

Step 2: Since Pre-register1 & Register is empty, the data in pre-register 2 is moved to Register automatically and executed instantly by ASIC.

Step 3: Then second function is called. CPU writes the corresponding values into pre-register2.

_8164_start_tr_move(0,20000.0,100.0,30000.0,0.0,0.2)

Step 4: Since Pre-register1 is empty; the data in pre-register 2 is moved to Pre-register1 automatically and wait to be executed.

Step 5: Now user can execute 3rd function, and it will be stored in Pre-register2

Step 6: When the first function finished, the Register becomes empty, and data in pre-register1 is allowed to move to register then executed instantly by ASIC, and, data in pre-register2 is moved to pre-register1.

Step 7: The ASIC will inform CPU by interrupt that motion is completed. And user can write 4th motion into Pre-Register 2.

Procedures to perform continuous motion

The following procedures are to help user making continuous motion.

Step 1:

(if Under Dos)

Enable the interrupt service by ***_8164*_int_contol()**

(if Under Windows)

Enable the interrupt service by ***_8164*_int_contol()** and ***_8164*_int_enable()**.

Step 2: Set bit "2" of INT factor to be "True" by ***_8164*_set_int_factor()**

Step 3: Set the "conti_logic" to be "1" by: ***_8164*_set_continuous_move()** *(note: if all motions are of relative mode, this function could be ignored.)*

Step 4: Call the first three motion functions.

Step 5: Wait for INT(under DOS) or EVENT(under Windows) of pre-register empty.

Step 6: call the 4th motion function.

Step 7: Wait for INT(if under DOS) / EVENT(if under Windows) of pre-register empty.

Step 8: call the 5th motion function.

(Repeat 7 , 8 ….continue….)

Step n: Call the last motion function and wait for all moves completion.

(note: Another way to detect completion of motion is by poling. User may constantly check the buffer status by **_8164_check_continuous_buffer()**.**)**

Restrictions of continuous motion

The statements below are restrictions and suggestions for continuous motion:

1. While Pre-register is not empty, user may not execute any more motion. Otherwise, the new one will overwrite the previous in pre-register2.
2. To get a continuity of velocity between 2 motions, the end velocity of previous and starting velocity of next must be the same. There are several methods to achieve this. The easiest way is to set the deceleration/acceleration time to be ‘ 0’ .

For example :

1st motion: *_8164_start_tr_move_XY(*0,1000,0,0,5000,0.2, 0.0*)*

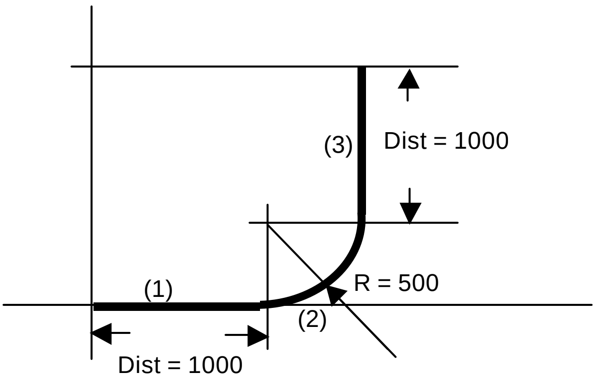
(Start a relative 2-axis linear interpolation, x distance=1000, y distance= 0 , start vel = 0, max vel = 5000, Tacc = 0.2, Tdec = 0)

2nd motion: *_8164_start_r_arc_xy*(0,0,500,500,500,1,5000);

(Start a relative 2-axis circular interpolation, center x distance=0, center y distance= 500 , End x distance = 500, end y distance = 500. max vel = 5000. It is a quarter ccw circle, with velocity = 5000)

3rd motion: *_8164_start_tr_move_XY(*0,0,1000,0,5000,0.0, 0.2*)*

(Start a relative 2-axis linear interpolation, x distance=0, y distance= 1000 , start vel = 0, max vel = 5000, Tacc = 0.0,Tdec = 0)



Explanation of example:

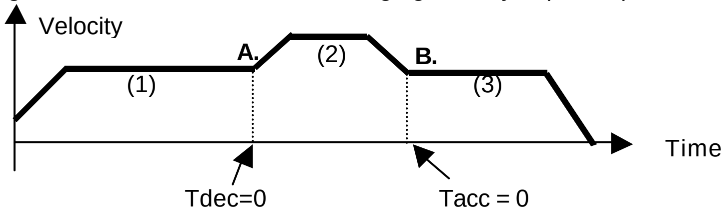
While these three motions were executed sequentially without waiting, the 1st occupies the Rigister and is executed instantly; the 2nd occupies Pre-Rigister 1and is waiting for completion of 1st; the 3rd occupies Pre-Rigister 2 and is waiting for completion of 2nd . Since the 1st motion has a ‘0’ deceleration time and 2nd is a arc of constant velocity, which is the as the max vel of the 1st, the PCI-8164 will output constant frequency at intersection between them.

1. Continuous motion between different axes is meaningless, for different axis get its own register and pre-register system.
2. Continuous motion between different number of axes is not allowed, for example: ***_8164_start_tr_move()*** can not be followed by ***_8164_start_ta_move_XY()* ,** vice versa, because these two functions belong to single axis and 2-axis mode individually.
3. It is possible to perform 3 axes or 4 axes continuous linear interpolation, but the speed continuity is impossible to achieve.
4. If any absolute mode is used during continuous motion, make sure that the ***_8164_reset_target_pos()*** is executed at least once after home move.(please refer to 4.1.8: Home return mode)

Examples of continuous motion

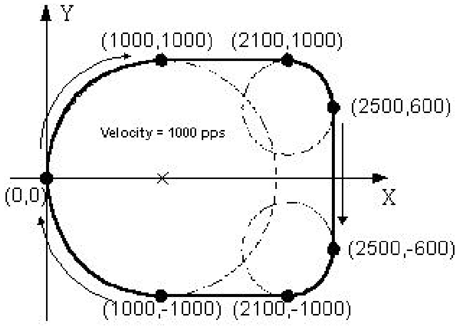
The following are example of continuous motion:

1. Single axes continuous motion: Changing velocity at preset point.



This example demonstrates how to use continuous motion function to achieve the velocity changing at pre-set point. The 1st motion (ta) moved axis to point A, with Tdec =0, and then the 2 nd continued instantly. The start velocity of (2) is the same with max velocity of (1), so that the velocity continuity exists at A. At point B. the Tacc of (3) is set to be 0, so the velocity continuity is also built.

2. 2-axis continuous interpolation:



This example demonstrates how to use continuous motion function to achieve 2-axis continuous interpolation. In this application, the velocity continuity is the key concern. Please refer to the example in previous page.

The functions related to continuous motion are listed below:

_*8164*_set_continuous_move(), _*8164*_check_continuous_buffer()

: Refer to section 6.17

4.1.8 Home Return Mode

In this mode, you can let the PCI-8164 output pulses until the condition to complete the home return is satisfied after writing the command **_8164_home_move()**. There are 13 home moving modes provided by PCI-8164. The "home_mode" of function **_8164_set_home_config()** is used to select one' s favorite.

After completion of home move, it is necessary to keep in mind that all the position related information should be reset to be " 0" . In PCI-8164, there are 4 counters and 1 software-maintained position recorder. They are :

Command position counter: To count the number of pulses output

Feedback position counter: To count the number of pulse input

Position error counter: To count the error between command and feedback pulse number.

General-Purposed counter: The source could be configured as pulse output, feedback pulse, manual pulser or CLK/2.

Target position recorder: To record the target position

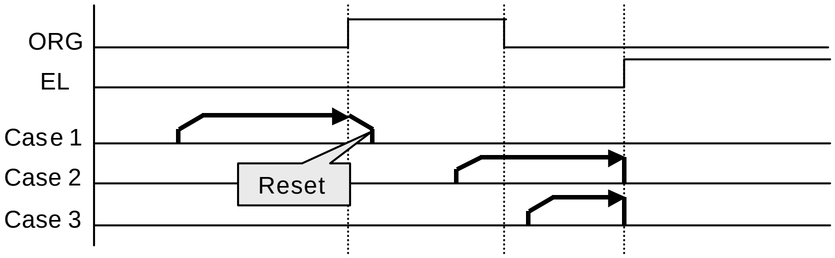
(Please refer to section 4.4 for more detail explanation about position counters)

After Home move complete, the first four counters will be cleared to " 0" automatically. However the **target position** recorder won' t. Because it is a software maintained, it is necessary to manually set the target position to " 0" by calling the function: **_8164_reset_target_pos().** So that, all the positions information will be " 0" .

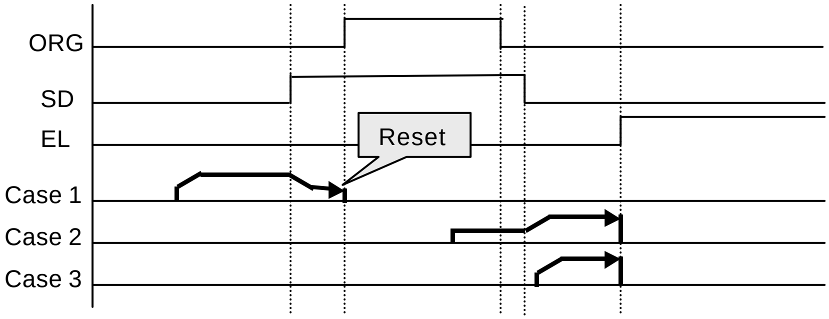
The following figures show the various home mode and the reset point, when the counter will be clear to " 0" .

home_mode=0: ORG → Slow down → Stop

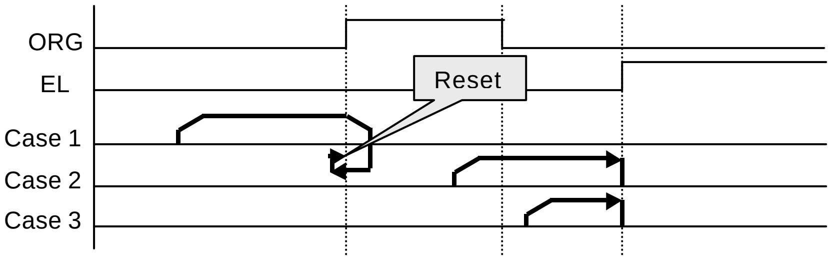
- When SD(Ramp-down signal) is inactive.



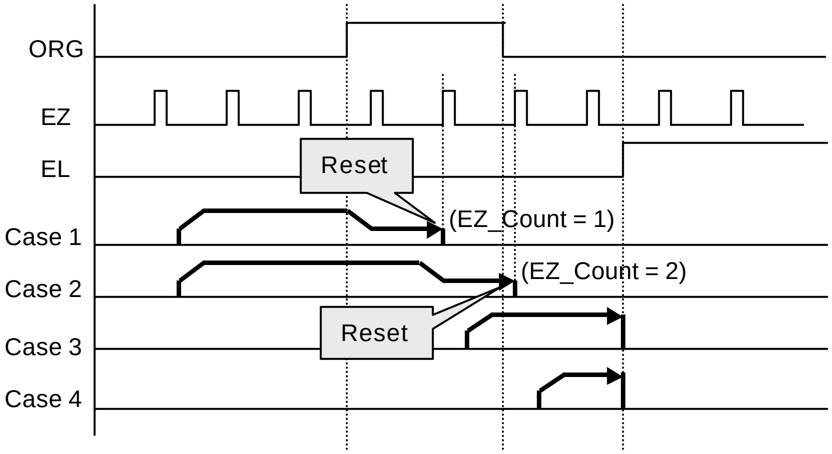
- When SD(Ramp-down signal) is active.



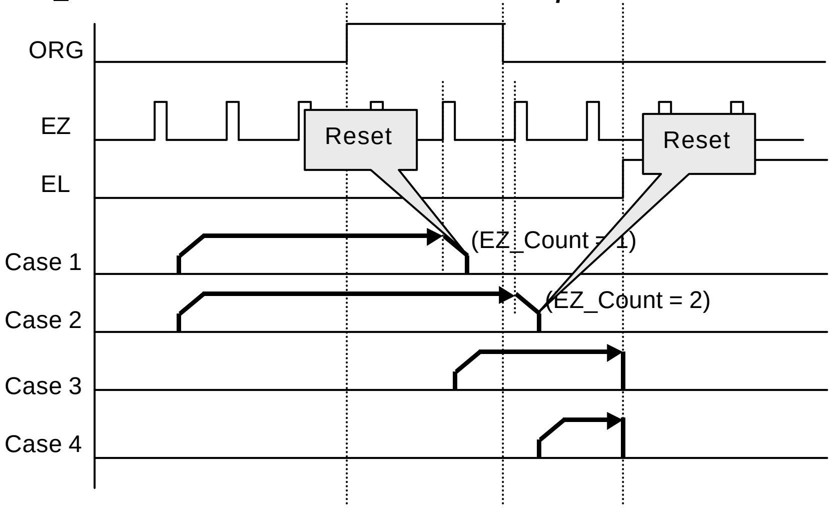
home_mode=1: ORG → Slow down → Stop at end of ORG



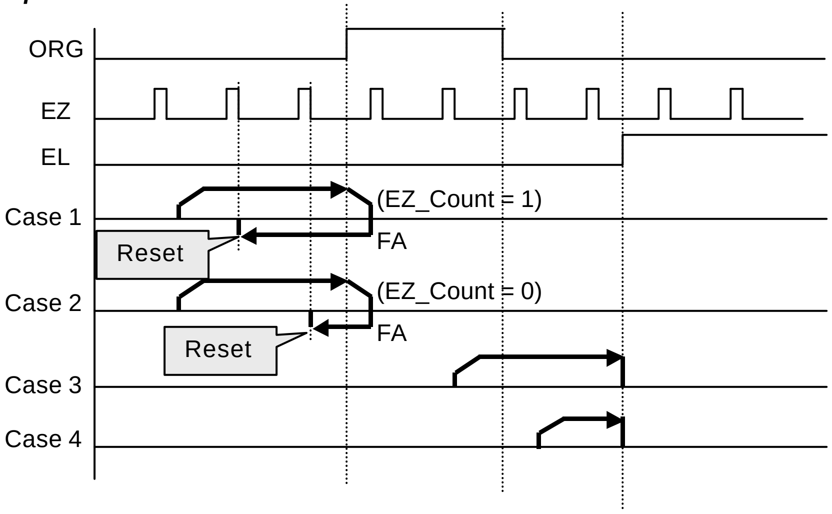
home_mode=2: ORG → Slow down → Stop on EZ signal



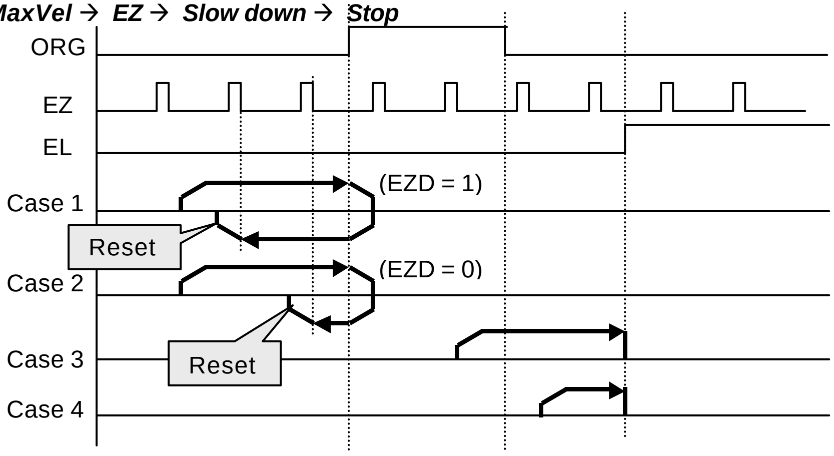
home_mode=3: ORG → EZ -→ Slow down → Stop



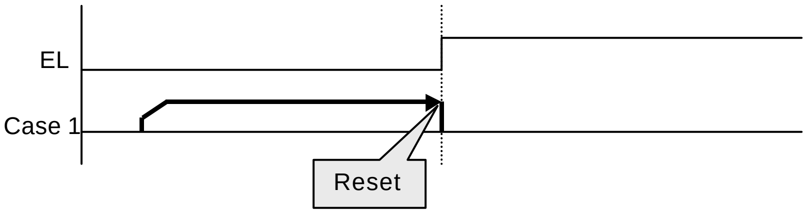
***home_mode=4**: ORG → Slow down → Go back at FA speed→ EZ→ Stop*



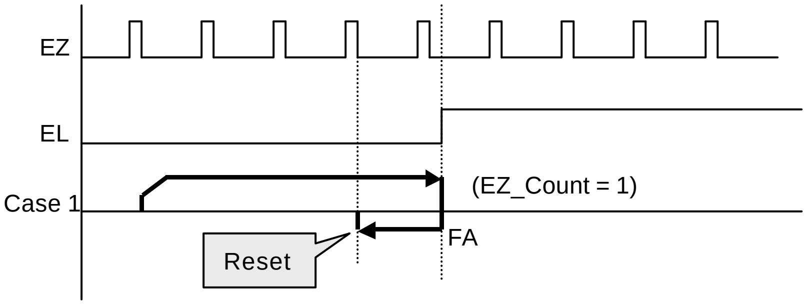
***home_mode=5**: ORG → Slow down → Go back → Accelerate to MaxVel → EZ → Slow down → Stop*



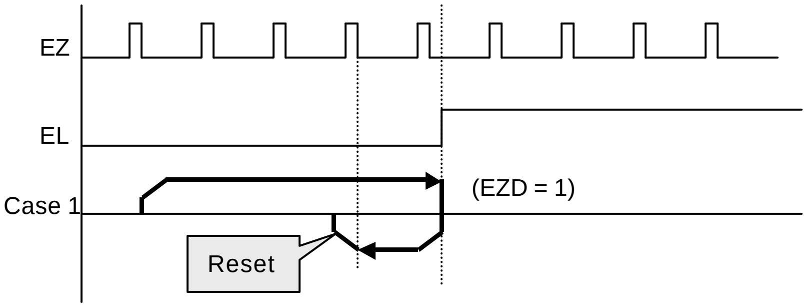
home_mode=6: *EL only*



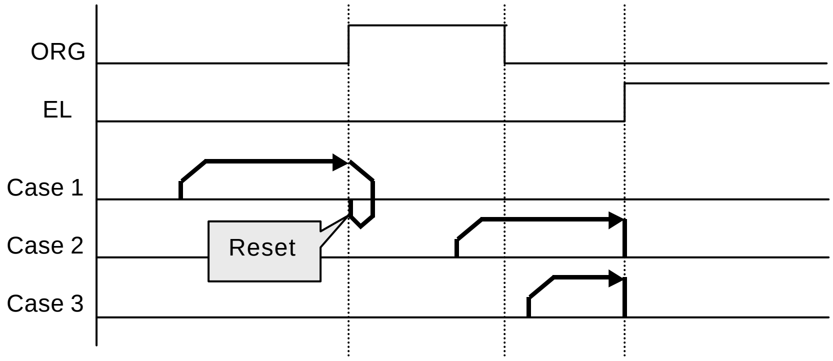
home_mode=7: EL → Go back → Stop on EZ signal



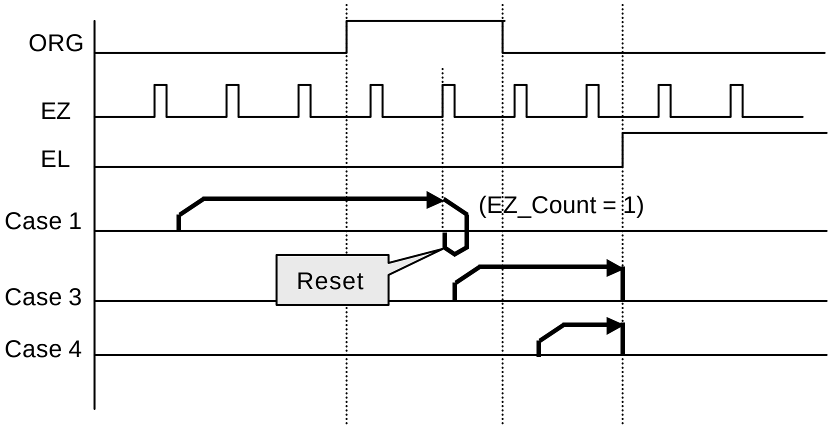
home_mode=8: EL → Go back → Accelerate to MaxVel → EZ → Slow down → Stop



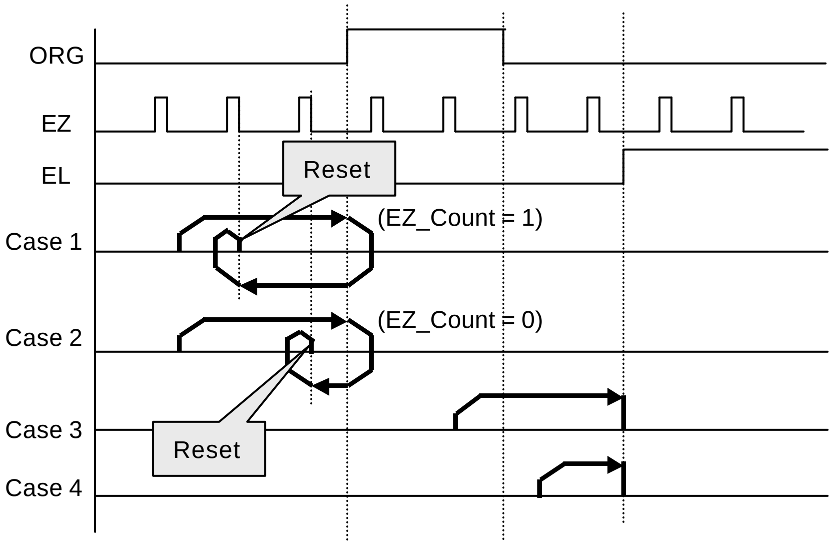
home_mode=9**: ORG → Slow down → Go back → Stop at beginning edge of ORG***



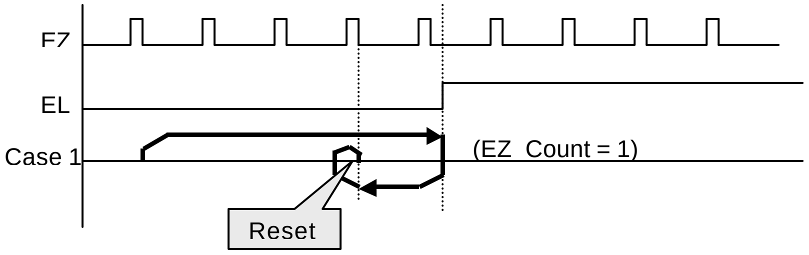
home_mode=10**: ORG → EZ → Slow down → Go back → Stop at beginning edge of EZ***



home_mode=11:* *ORG → Slow down → Go back (backward)→ Accelerate to MaxVel→ EZ→ Slow down → Go back again (forward)→ Stop at beginning edge of EZ



home_mode=12:* *EL → Stop → Go back (backward)→ Accelerate to MaxVel → EZ → Slow down → Go back again (forward)→ Stop at beginning edge of EZ



Relative Functions:

_8164_set_home_config(), _8164_home_move() : ***Refer to section 6.9***

4.1.9 Manual Pulser Mode

For manual operation of a device, you may use a manual pulser such as a rotary encoder. The PCI-8164 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 when a **_8164_pulser_vmove(), _8164_pulser_pmove()** or **_8164_pulser_home_move()** command has been called. To stop, by a **_8164_sd_stop()** or**_8164_emg_stop()** command or till satisfaction of movement.

The PCI-8164 receives plus and minus pulses (CW/CCW) or 90 degrees phase difference signals(AB phase) from the pulser at PA and PB pins. To set the input signal modes of pulser, use **_8164_set_pulser_iptmode()** function. The 90° phase difference signals can be input through multiplication by 1, 2 or 4. If the AB phase 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 leading the PB signal by 90° phase.

Relative Functions:

_8164_pulser_vmove(), _8164_pulser_pmove(), _8164_pulser_home_move()

_8164_set_pulser_iptmode(): Refer to section 6.10

4.2 The motor driver interface

The PCI-8164 provides the INP, ALM, ERC, SVON, RDY 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. The SVON is general-purposed output signal, and RDY is general-purposed input signal. The meaning of "general-purposed" is that the processing of signal is not a build-in procedure of hardware. The hardware processes INP, ALM and ERC signals according to pre-defined rule. For example, when receiving ALM signal, the PCI-8164 stop or decelerate to stop output pulses automatically. However, SVON and RDY are not that case, they actually act like common I/O.

4.2.1 INP

The processing of INP signal is a hardware build-in procedure, and it is designed to cooperate with the in-position signal of servomotor driver.

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. Then, the servo driver sends out the in-position signal (INP) to the pulse generator to indicate the motor stops running.

Usually, the PCI-8164 stops outputting pulses upon completion of outputting designated pulses. But by setting parameter ***inp_enable*** in ***_8164_set_inp()*** function, you can delay the completion of motion to the time when the INP signal is turned on, ie, the motor arrives the target position. Status of **_8164_motion_done()** and INT signal are also delayed. That is, when performing under position control mode, the completion of **_8164_start_ta_move()**, **_8164_start_sr_move()**,…etc, is delayed until INP signal is turned ON.

The in-position function can be enable or disable, and the input logic polarity is also programmable by parameter " **inp_logic"** of ***_8164_set_inp()***. The INP signal status can be monitored by software function: **_8164_get_io_status()**.

Relative Functions:

_8164_set_inp()* : *Refer to section 6.12
_8164_get_io_status(): ***Refer to section 6.13***
_8164_motion_done(): ***Refer to section 6.11***

4.2.2 ALM

The processing of ALM signal is a hardware build-in procedure, and it is designed to cooperate with the alarm signal of servomotor driver.

The ALM signal is an output from servomotor driver. Usually, It is designed to inform that something wrong with the driver or motor.

The ALM pin receives the alarm signal output from the servo driver. The signal immediately stops the PCI-8164 from generating pulses or stops it after deceleration. If the ALM signal is in the ON status at the start, the PCI-8164 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 microseconds.

You can change the input logic of ALM by set the parameter " **alm_logic**" of ***_8164_set_alm*** function and the stop mode by "**alm_mode**". Whether or not the PCI-8164 is generating pulses, the ALM signal lets it output the INT signal.. The ALM status can be monitored by software function: ***_8164_get_io_status()***. The ALM signal can generate IRQ if the interrupt service is enabled, please refer to section 4.7.

Relative Functions:

_8164_set_alm(): Refer to section 6.12
_8164_get_io_status(): ***Refer to section 6.13***

4.2.3 ERC

The ERC signal is an output from PCI-8164. The processing of ERC signal is a hardware build-in procedure, and it is designed to cooperate with the deviation counter clear signal of servomotor driver.

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-8164, it keeps moving till the deviation counter of the driver down to zero even if the PCI-8164 stop outputting pulses because of the ±EL signal or the completion of home return. 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 pulse width is a time length defined by function call ***_8164_set_erc()***. The ERC signal will automatically output when ±EL signals, ALM signal is turned on to immediately stop the servomotor.

Relative Functions:

_8164_set_erc()* : *Refer to section 6.12

4.2.4 SVON and RDY

In PCI-8164, every axis is equipped with SVON and RDY, which are general-purposed output and input channels, respectively. Usually, the SVON is useful to cooperate with servomotor drivers as Servo ON command, and RDY to receive the Servo Ready signal from servomotor drivers. That is the reason why they are named as SVON and RDY. There is no build-in procedure for SVON and RDY.

The SVON signals are controlled by software function: **_8164_Set_Servo()**.

RDY pins are dedicated for digital input use. The status of this signal can be monitored by software function **_8164_get_io_status()**.

Relative Functions:

_8164_Set_Servo(): ***Refer to section 6.12***

_8164_get_io_status(): ***Refer to section 6.13***

4.3 The limit switch interface and I/O status

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

- SD/PCS: Ramping Down & Position Change sensor
- ±EL: End-limit sensor
- ORG: Origin position

In any operation mode, if an ±EL signal is active during moving condition, it will cause PCI-8164 to stop output pulses automatically. If an SD signal is active during moving condition, it will cause PCI-8164 to decelerate. If operating in multi-axes mode, it automatically applied to all related axes.

4.3.1 SD/PCS

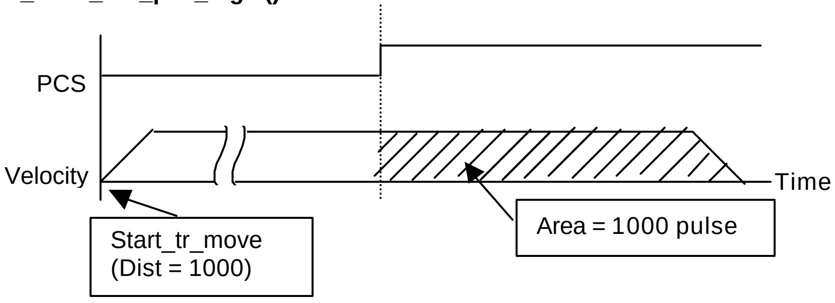
The SD/PCS pin for each axis is an input channel and is selectable to connect into SD (Slow Down) or Position Change Signal(PCS). To configure it, by function call **_8164_set_sd_pin()**.

When SD/PCS pin is directed to SD (the default setting), the PCS signal will be kept in low level. And, while as PCS is selected, the SD signal will be kept in low level. Users need to take care the logic and enable/disable attributes for signal not used.

The slow-down signals are used to force the output pulse (OUT and DIR) to decelerate to and then keep on the StrVel when it is active. The StrVel is usually smaller than MaxVel, so, this signal is very useful to protect the mechanism moving under high speed toward the mechanism limit. SD signal is effective for both plus and minus directions.

The ramping-down function can be enable or disable by software function: **_8164_set_sd()**. 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 **_8164_get_io_status()**.

The PCS signal is used to define the starting point of a preset tr, sr motion. Refer to the following chart. The logic of PCS is configurable by ***_8164*_set_pcs_logic()**



Relative Functions:

_8164_set_sd_pin(),*_8164_*set_pcs_logic(): ***Refer to section 6.5***
_8164_set_sd(): Refer to section 6.12
_8164_get_io_status(): ***Refer to section 6.13***

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. There are two possible stop modes, one is "stop immediately", and, the other is "decelerate to StrVel then stop". To select the mode: **_8164_set_el().**

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, only the negative (minus) direction output pulse can be generated for moving the motor to negative (minus) direction.

The EL signal can generate IRQ if the interrupt service is enabled, please refer to section 4.7.

You can use either 'a' contact switch or 'b' contact switch by setting the dip switch S1. The PCI-8164 is delivered from the factory with all bits of S1 set to OFF.

The signal status can be monitored by software function: **_8164_get_io_status().**

Relative Functions:

_8164_set_el(): ***Refer to section 6.12***

_8164_get_io_status(): ***Refer to section 6.13***

4.3.3 ORG

The ORG signal is used, when the motion controller is operated at the home return mode. There are 13 home return modes (please refer to section 4.1.8), you can select one of them by setting "*home_mode*" argument in software function: **_8164_set_home_config()**. The logic polarity of the ORG signal, level input or latched input mode are selectable by this software function.

After setting the configuration of home return mode by **_8164_set_home_config()**, a **_8164_home_move()** command can perform the home return function.

Relative Functions:

_8164_set_home_config(),_8164_home_move(): ***Refer to section 6.19***

4.4 The Counters

The PCI-8164 provides 4 counters for every axis, and, they are introduced in this section:

Command position counter: To count the number of pulses output

Feedback position counter: To count the number of pulse input

Position error counter: To count the error between command and feedback pulse number.

General-Purposed counter: The source could be configured as pulse output, feedback pulse, manual pulser or CLK/2.

Also, the **target position recorder**, a software-maintained position recorder, is discussed.

4.4.1 Command position counter

The command position counter is a 28-bits binary up/down counter, and its input source is the output pulse from PCI-8164, thus, it provide as exact information of current command position. Note: the command position is different from target position. The **command position** increases or decreases as pulse output, while the **target position** changes only when a new motion command was executed. The target position is recorded by software, and need manually resetting after home move completed.

The command position counter will be clear to "0" automatically after home move completed. Besides, the function call, **_*8164*_set_command()**, can be executed in any time to set an new command position value. To read current command position: **_*8164*_get_command().**

Relative Functions:

_*8164*_set_command(),_*8164*_get_command(): ***Refer to section 6.15***

4.4.2 Feedback position counter

The PCI-8164 has a 28-bits binary up/down counter for managing the present position feedback 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 **_8164_*set_pls_iptmode()*** function.

In case that some applications don' t implement an encoder, it is possible to set the feedback counter source to be the output pulse, just as the command counter. Thus, the feedback counter and the command counter will actually have the same value. To enable the counters counting pulses input from pulse output, set "***Src***" parameter of software function ***_8164_*set_feedback_src*()*** to " 1" .

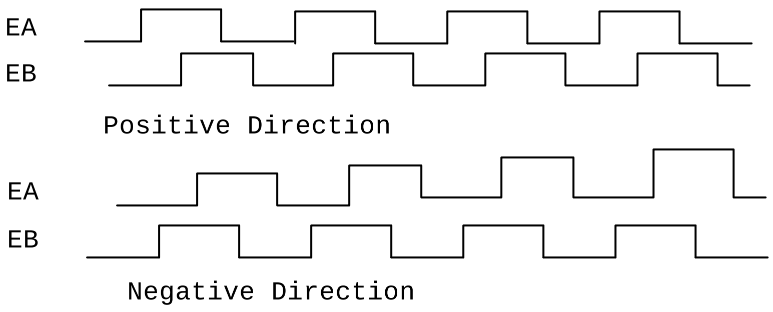
Plus and Minus Pulses Input Mode(CW/CCW Mode)

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 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 pos ition of the mechanism. The input logic polarity of the EZ signals is programmable by software function **_8164_*set_home_config().*** The EZ signals status of the four axis can be monitored by ***get_io_status()***.

The feedback position counter will be clear to “0” automatically after home move completed. Besides, the function call, ***_8164*_set_position()**, can be executed in any time to set an new command position value. To read current command position: ***_8164*_get_position().**

Relative Function:

_8164_set_pls_iptmode(), _8164_set_feedback_src() :Refer to section 6.4

***_8164*_set_position(), *_8164*_get_position(): Refer to section 6.15**

_8164_set_home_config(): Refer to section 6.9

4.4.3 Position error counter

The position error counter is used to calculate the error between command position and feedback position. The working theory is that it adds one count when PCI-8164 output one pulse and subtracts one count when PCI-8164 receives one pulse (from EA,EB). It is very useful to detect the step-losing situation (stall) of stepping motors when encoder is applied.

Since the position error counter automatically calculate the difference between pulse output and pulse feedback, it is inevitable to get error if the motion ratio is not equal to “1”.

To get the position error, use the function call: **_8164_g*et_error_counter().*** To reset the position error counter, use the function call: **_8164_res*et_error_counter().*** The position error counter will automatically clear to “0” after home move complete.

Relative Function:

_8164_get_error_counter(),_8164_reset_error_counter() :Refer to section 6.15

4.4.4 General-Purposed counter

The source of general-purposed counter is the most versatile, it could be:

1. Pulse output - just as command position counter
2. Pulse input – just as feedback position counter
3. **Manual Pulser inpu**t – the **default status.**
4. Clock – an accurate timer. (9.8 MHz)

The default source of general-purposed counter is manual pulser. (Please refer to section 4.1.9 for detail explanation of manual pulser) To set other source, use the function call: ***_8164_*****set_general_counter().**To get the counter value, use the function call: ***_8164_*****get_general_counter().**

Relative Function:

_8164_**set_general_counter(),** ***_8164_*****get_general_counter() : Refer to section 6.15**

Counter	Description	Counter Source	Function	Function description
Command position	To count the number of pulses output	pulses output	*_8164*_set_command	Set a new value for command position
			*_8164*_get_command	Read current command position:
Feedback position	To count the number of pulse input	**EA/EB** or pulse output	_8164_set_pls_iptmode	Select the input modes of EA/EB
			_8164_set_feedback_src	Set the counters input source
			*_8164*_set_position	Set a new value for feedback position
			*_8164*_get_position	Read current feedback position:
Position error	To count the error between command and feedback pulse	EA/EB and pulse output	_8164_get_error_counter	To get the position error, use the function call:
			_8164_reset_error_counter	To reset the position error counter
General-Purposed	General-purposed counter	Pulse output EA/EB **manual pulser** CLK/2.	*_8164*_set_general_counter	Set a new counter value
			*_8164*_get_general_counter	Read current counter value

4.4.5 Target position recorder

The target position recorder is very useful for providing target position information. For example, if the PCI-8164 is operating in continuous motion with absolute mode, the target position let next absolute motion knows the target position of previous one.

It is very important to know that the target position recorder is handled by software. Every time when a new motion command is executed, the displacement is added automatically into the target position recorder. To make sure the correctness of target position recorder, user needs to manually maintain it in the following two situations by the function call **_8164_reset_target_pos()**:

1. After Home move complete
2. After new feedback position is set

Relative Functions:

_8164_reset_target_pos(): ***Refer to section 6.15***

4.5 Multiple PCI-8164 Cards Operation

The software function library support maximum up to 12 PCI-8164 Cards, that means maximum up to 48 axes of motors can be controlled. Since PCI-8164 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-8164 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-8164 when multiple cards are applied. The card number of one PCI-8164 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 effect 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-8164 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:

```
_8164_start_tv_move(6, 0, 10000, 0.5);
```

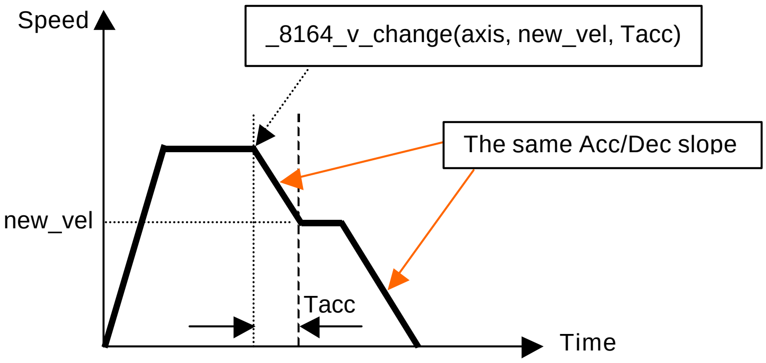
To determine the right card number, Try and Error may be necessary before application. Motion Creator can be utilized to minimize the search time.

For applications needed to move many axes simultaneously on multiple PCI_8164 cards, users should follow the connection diagrams in Section 3.12 to make connections between their CN4 connectors. Several functions illustrated in Section 6.8 may be useful when writing programs for such applications.

4.6 Change position or speed on the fly

The PCI-8164 provides powerful position or speed changing function while axis is moving. Changing speed/position on the fly means that the target speed/position can be altered after the motion started. Yet, these functions are not unlimited. Please study carefully all constrains before implement on-the-fly function.

4.6.1 Change speed on the fly



The change speed on the fly function is applicable on single axis motion only. Both velocity mode motion and position mode motion is applicable. The graph above shows the basic operating theory.

The following functions are related to change speed on the fly function.

_8164_v_change() – change the MaxVel on the fly
_8164_cmp_v_change() –change velocity when general comparator comes into existence
_8164_sd_stop() – slow down to stop
_8164_emg_stop() – immediately stop
_8164_fix_speed_range() – define the speed range
_8164_unfix_speed_range() – release the speed range constrain

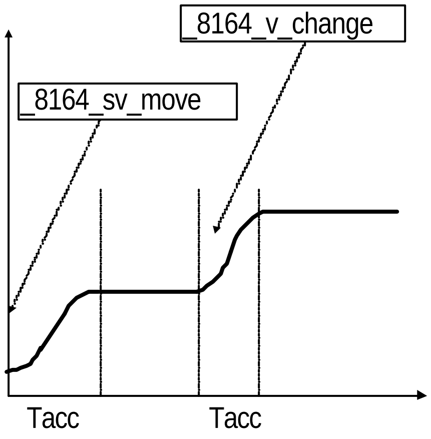
All the first 4 functions can do the speed changing during single axis motion. However, the **_8164_sd_stop()** and **_8164_emg_stop() only** change the axis speed to "0". The **_8164_fix_speed_range()** is necessary before any **_8164_v_change()** function, and **_8164_unfix_speed_range()** release the speed range constrained by **_8164_fix_speed_range()**.

The **_8164_cmp_v_change()** gets almost the same function as **_8164_v_change()**, except that the **_8164_cmp_v_change()** act only when general comparator comes into existence. Please refer to section 4.4.4 for more detail description about general comparator.

The last 4 functions are relatively easy to understand and use. So, the discussion will be focused on the **_8164_v_change()**

Work theory of _8164_v_change() :

The _8164_v_change() function is used to change the MaxVel on the fly. In a normal motion operation, the axis starts at StrVel speed, accelerates to MaxVel, and then keeps at MaxVel until entering deceleration region. If user changes the MaxVel, it will force the axis to accelerate or decelerate to a new speed in a period of time defined by user. Both Trapezoidal and S-curve profiles are applicable. The speed changing is at constant acceleration in Trapezoidal profile, and constant jerk in S-curve.



Speed change with S-Curve

Constrains of _8164_v_change():

In single axis preset mode, there must be enough remaining pulses to reach new velocity. If not, the _8164_v_change() will return error and keep velocity unchanged.

For example:

A trapezoidal relative motion is applied:

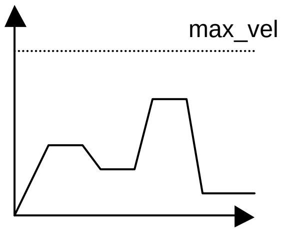
_8164_start_tr_move(0,10000,0,1000,0.1,0.1).

It cause axis 0 to move for 10000 pulse, and the maximum velocity is 1000 PPS.

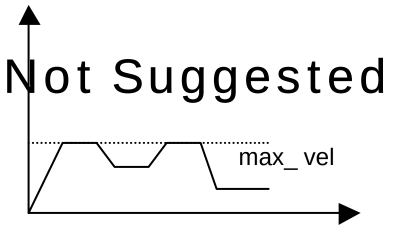
At 5000 pulse the _8164_v_change(0,NewVel,Tacc) is applied.

NewVel (PPS)	Tacc (Sec)	Necessary remaining pulses			OK / Error
		Acceleration	Deceleration	Total	
5000	0.1	300	313	613	OK
5000	1	3000	3125	6125	Error
10000	0.1	550	556	1106	OK
50000	0.1	2550	2551	5101	Error

1. User must set the maximum velocity by **_8164_fix_speed_range()** so that the **_8164_v_change()** could work correctly. If not, the MaxVel set by **_8164_v_move()** or **_8164_start_ta_move**() becomes automatically the maximum velocity which **_8164_v_change()** could not exceed.

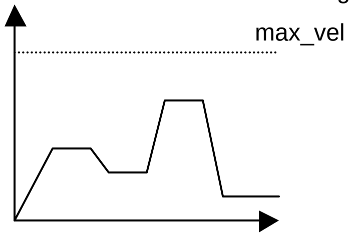


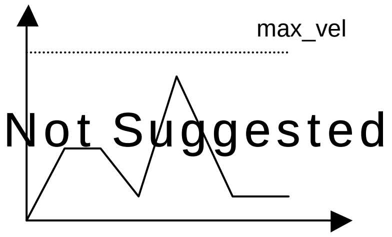
With **fix_speed_range**



Without **fix_speed_range**

2. **_8164_v_change()** during acceleration or deceleration period is not suggested. Though it does work in most cases, the acceleration and deceleration time is not guaranteed.

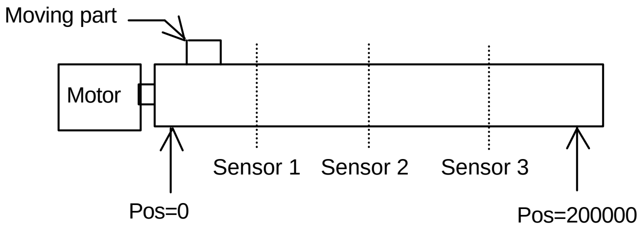




Example:

There are 3 speed change sensors 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.

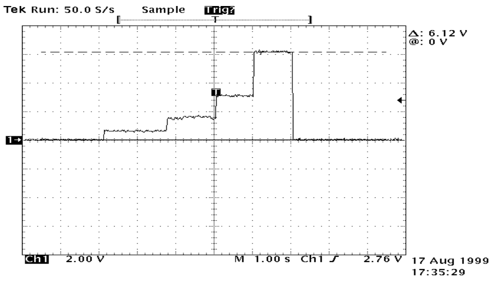


```
#include " pci_8164.h"

_8164_fix_speed_range(axis, 100000.0);
_8164_start_ta_move(axis, 200000.0, 1000, 10000, 0.02,0.01);
while(!_8164_motion_done(axis))
  {
                    // Get Sensor' s information from other I/O card

        if((Sensor1==High) && (Sensor2==Low) && (Sensor3 == Low))
                              _8164_v_change(axis, 25000, 0.02);
        else if((Sensor1==Low) && (Sensor2==High) && (Sensor3 == Low))
                              _8164_v_change(axis, 50000, 0.02);
        else if((Sensor1==Low) && (Sensor2==Low) && (Sensor3 == High))
                              _8164_v_change(axis, 100000, 0.02);
  }
```

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

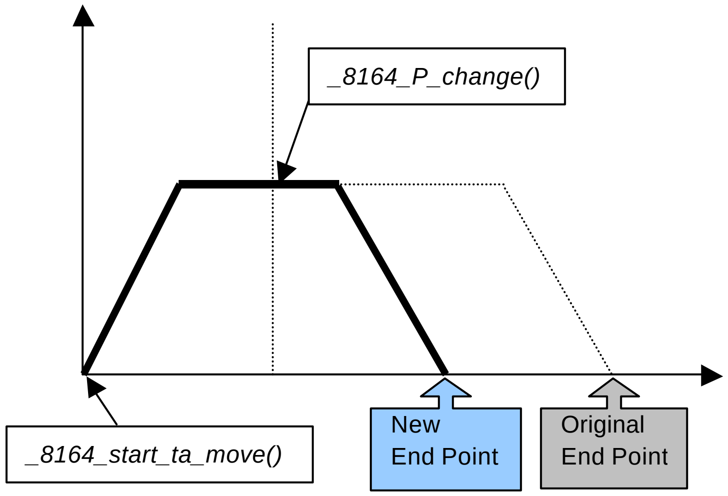


Relative Function:

_8164_v_change(), _8164_sd_stop(), _8164_emg_stop()
_8164_fix_speed_range(),_8164_unfix_speed_range()
_8164_*get*_currebt_speed()
: refer to section 6.5

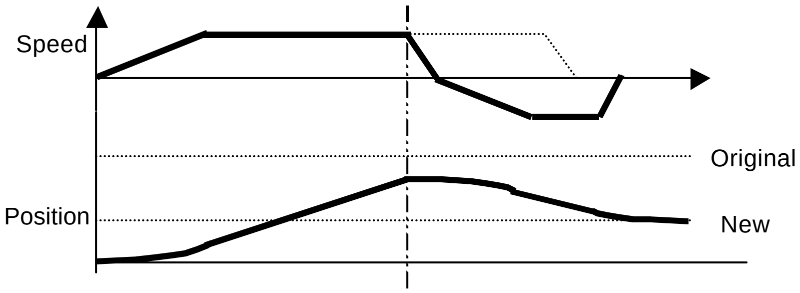
4.6.2 Change position on the fly

When operating in single-axis absolute pre-set motion, it is possible to change the target position during moving by function call **_8164_p_change().**



Work theory of _8164_p_change() :

The _8164_p_change() is applicable on _8164_start_ta_move(), and _8164_start_sa_move() only. It is to change the target position defined originally by these two functions. After changing position, the axis will move to the new target position and totally forget the original position. If the new position is in passed path, it will cause the axis to decelerate to stop, than reverse, as the following graph. The acceleration and deceleration rate, the StrVel and MaxVel will keep the same as original setting.



Constrains of _8164_p_change() :

1. **_8164_p_change()** is only applicable on single-axis absolute pre-set motion, ie, **_8164_start_ta_move()**, and **_8164_start_sa_move()** only
2. Position change on deceleration period is not allowed.
3. There must be enough distance between new target position and current position where **_8164_p_change()** is executed. Because, PCI-8164 needs enough space to finish deceleration.

For example:

A trapezoidal absolute motion is applied:

*_8164_start_ta_move(0,**10000**,0,1000,0.5,1).*

It cause axis 0 to move to pulse 10000 position, and the maximum velocity is 1000 PPS. The necessary number of pulses to decelerate is 0.5*1000*1 = 500.

At position " CurrentPos" the _8164_p_change(0, NewPos) is applied.

NewPos	CurrentPos	OK / Error	Note
5000	4000	OK	
5000	4501	Error	
5000	5000	Error	
5000	5499	Error	
5000	6000	OK	Go back
5000	9499	OK	Go back
5000	9500	Error	
5000	9999	Error	

Relative Function:

_8164_p_change() : refer to section 6.6

4.7 Position compare and Latch

The PCI-8164 provides position compare function in axis 0 and 1,and position latch function in axis 2 and 3. The compare function is to "output a trigger pulse when counter reached the value set by user". CMP1 (Axis0) and CMP2 (Axis 1) are used for compare trigger. The latch function is to capture values of all 4 counters (refer to section 4.4) at that instant latch signal activate. LTC3 (Axis 2) and LTC4 (Axis 3) are used to receive latch pulse.

4.7.1 Comparators of PCI-8164

There are 5 comparators in every axis of PCI-8164. Each comparator gets its unique functionality. Here is the table of description:

	Compare Source	Description	Function Related
Comparator 1	Command position counter	Soft Limit (+) **(Refer to section 4.9)**	*_8164_set_*softlimit *_8164*_enable_softlimit *_8164*_diable_softlimit
Comparator 2	Command position counter	Soft Limit (-) **(Refer to section 4.9)**	
Comparator 3	Position error counter	Step-losing detection	*_8164*_error_counter_*check*
Comparator 4	Any counters	General- purposed	*_8164*_set_general_*comparator*
Comparator 5 (Only Axis 0 & 1)	Feedback position counter	Position compare function (Trigger)	*_8164*_set_trigger_comparator *_8164*_build_compare_function *_8164*_build_compare_table *_8164*_set_auto_compare

Note: Not all the 5 comparator get the ability to trigger output pulse via CMP. It is only the comparator 5.

The compare 1 & 2 are for soft limit, please refer to section 4.9. The comparator 3 is used to compare with position error counter. It is very useful for detecting if a stepping motor lost pulses. To enable/disable the step-losing detection and set the allowed tolerance: ***_8164_set_error_counter_check()***

The PCI-8164 will generate an interrupt if step-losing is enabled and occurred.

The comparator 4 is a general-purposed comparator, which will generate interrupt (default reaction) if the comparing condition comes into existence. The comparing source counter can be any counter. The compared value, source counter, comparing method and reaction are set by software function call ***_8164*_set_general_*comparator()*** .

4.7.2 Position compare

The position compare function is performed by the 5th comparator, whose comparing source is the feedback position counter. Only the first 2 axes (0 and 1) can do position compare function. The position compare function is to trigger a pulse output via CMP, when the comparing condition comes into existence.

The comparing condition consists of 2 parts, the first is the value to be compared, and the second is comparing mode. Comparing mode can be “>“ , “=“ or “<“. The easiest way to use position comparison function is to call the software function:

_*8164*_set_trigger_comparator(AxisNo, Method, Data)

The second parameter “Method” indicates the comparing method, while the third “Data” is for value to be compared. In continuous comparing, this data will be ignored automatically since the compare data will be build by other functions.

Continuously compare

For user who wants to compare multiple data continuously, functions of building comparison tables is also provided as shown in the following

Under DOS (always use FIFO)

1. *_8164_build_comp_function(AxisNo, Start, End, Interval)*
2. *_8164_build_comp_table(AxisNo, tableArray, Size)*
3. *_8164_set_auto_compare(AxisNo, SelectSource)*

Under Windows (FIFO & RAM selectable)

1. *_8164_build_comp_function(AxisNo, Start, End, Interval, device)*
2. *_8164_build_comp_table(AxisNo, tableArray, Size, device)*
3. *_8164_set_auto_compare(AxisNo, SelectSource)*

While under windows, the parameter “Deice” is to indicate if FIFO is used to store compare data. If Device = 1, FIFO is used. If Device = 0, the RAM is used. The difference is that RAM needs CPU to cover the reloading job and interrupt is necessary to notify reloading. When FIFO is used, the reloading is made by hardware.

Note: Please turn off all interrupt function, when FIFO is used.

The first function is to build a compare list by start point, end point and constant interval. The second is to build an arbitrary comparing table (data array). The difference between them is that function has no limitation in size of comparing points in RAM mode, while the table array size is limited to

1024 points. But, if FIFO is used, the maximum size of data is always 4096 no matter function or table is used.

The third function is a comparing source selection function.

If SelectSource = 0, continuously comparing functionality is not used.

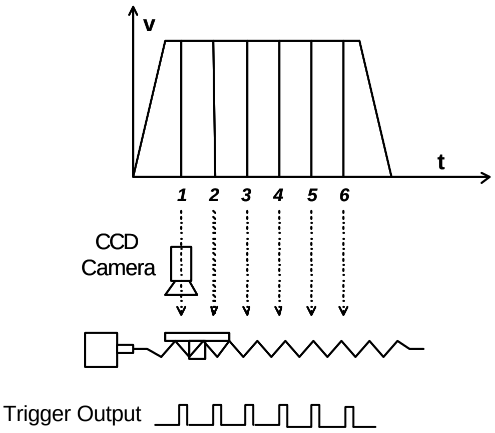
If SelectSource = 1, use FIFO.

If SelectSource = 2, use function (**RAM, window only**).

If SelectSource = 3, use table (**RAM, windows only**).

To check current compare data: **_8164_check_compare_data():**

Here is an example of using continuous position comparison functions.



In this application the table is controlled by the motion command and the CCD Camera is controlled by the position comparison output of PCI-8164. The image of moving object can be got in this way easily.

The example code is shown in the following

(Under Dos)

```
_8164_set_trigger_comparator(0, 2, 10000);  // Set comparing method
                                            // 2 : for “ feedback > data”
for( i=0 ; i<6 ; i++)
        CompTable[i] = 10000 + 10000 * i;       // Build Compare Data
array
_8164_Build_Comp_Table(0, CompTable, 6); // Set compare table
_8164_set_auto_compare(0, 1,1);             // Select comparing source
```

```
                                                  // FIFO is used
_8164_start_r_move( Axis0, 80000, 0,10000, 0.5);
```

(Under Windows)

```
_8164_set_trigger_comparator(0, 2, 10000);  // Set comparing method
                                            // 2 : for " feedback > data"
for( i=0 ; i<6 ; i++)
    CompTable[i] = 10000 + 10000 * i;       // Build Compare Data array
_8164_Build_Comp_Table(0, CompTable, 6,1); // Set compare table
_8164_set_auto_compare(0, 1);               // Select comparing source
                                            // FIFO is used
_8164_start_r_move( Axis0, 80000, 0,10000, 0.5);
```

The following guidelines will be of much help in using continuous compare.

1. ***Decides the comparing mode***:
   Use **_*8164*_set_trigger_comparator()** function.
2. ***Sets up the desired comparison data***: There are two functions:
   (under Dos)
   _8164_build_comp_function(AxisNo, Start, End, Interval)
   _8164_build_comp_table(AxisNo, tableArray, Size)
   (under Windows)
   _8164_build_comp_function(AxisNo, Start, End, Interval, Device)
   _8164_build_comp_table(AxisNo, tableArray, Size, Device)
3. ***Sets up the comparing source***:
   _8164_set_auto_compare(AxisNo, SelectSource)

Relative Function:

_*8164*_set_trigger_comparator(), _8164_build_comp_function()
_8164_build_comp_table(), _8164_set_auto_compare()
_8164_check_compare_data(),_8164_set_trigger_type ()
: refer to section 6.16

4.7.3 Position Latch

Position latch is a contrary function to position compare. The position compare function is to trigger a pulse output via CMP, when the comparing condition comes into existence. Yet, the position latch function is to receive pulse input via LTC, and then capture all counters' (refer to section 4.4) data in that instant. The latency between occurring of latch signal and finishing of position capturing is extremely short, for the latching procedure is made by hardware. Only the last 2 axes (2 and 3) can do position latch function. LTC3 (Axis 2) and LTC4 (Axis 3) are used to receive latch pulse.

To set the latch logic: **_8164_set_ltc_logic().**

To get the latched values of counters: **_8164_get_latch_data(AxisNo, CntNo, Pos).** The second parameter "CntNo" is used to indicate the counter of which the latched data will be read.

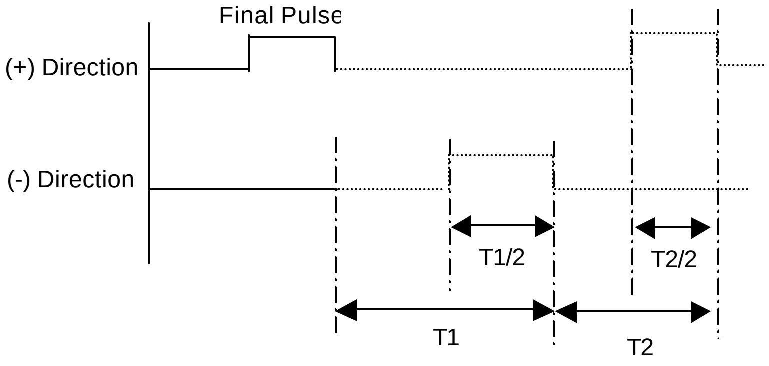
Relative Function:

_8164_set_ltc_logic(),_8164_get_latch_data() : refer to section 6.16

4.8 Hardware backlash compensator and vibration suppression

Whenever direction change is occurred, The PCI-8164 outputs backlash corrective pulses before sending commands. The function **_8164_backlash_comp()** is used to set the pulse number.

In order to minimize vibration when a motor stops. The PCI-8164 can output single pulse for negative direction and then single pulse for positive direction right after completion of command movement. Refer to following figure, the function **_8164_suppress_vibration()** is used to set the T1 & T2.



Relative Function:

_8164_backlash_comp(),_8164_suppress_vibration()
: refer to section 6.6

4.9 Software Limit Function

The PCI-8164 provides 2 software limits for each axis. The soft limit is extremely useful to protect user' s mechanical system, for it works as a physical limit switch, when setting correctly.

The soft limit is built on comparator 1 and 2 (please refer to section 4.7.1), and the comparing source is command position counter. The working theory is that pre-setting limits value on comparator 1 and 2, then, when the command position counter reached the limit value, the PCI-8164 reacts as the physical limit switch is touched. Thus, it stops immediately or decelerates to stop pulse output.

To set the soft limit: ***_8164_set_*****softlimit();**
To enable soft limit: ***_8164_*****enable_softlimit();**
To disable soft limit: ***_8164_*****diable_softlimit()**

Note: The soft limit is applied to **command position**, but not the **feedback position** (please refer to 4.4). In case the moving ratio is not equal to " 1" , it is necessary for user to manually calculate the corresponding command position where the soft limit is, when using **_8164_set_softlimit()**.

Relative Function:

_8164_set_**softlimit(),*****_8164_*****enable_softlimit(), *****_8164_*****diable_softlimit()**
: refer to section 6.16

4.10 Interrupt Control

The PCI-8164 motion controller can generate INT signal to host PC. The parameter "**intFlag**" of software function call **_8164_int_control(),** can enable/disable the interrupt service.

After a interrupt occurred, the function _8164_get_int_status() is used to receive the INT status, which contains information about INT signal. The int status of PCI-8164 is composed of two independent parts: **error_int_status** and **event_int_status**. The event_*int*_status recodes the motion and comparator event under **normal operation**, and this kind of INT status can be masked by **_8164_set_int_factor()**. The error_int_status is for abnormal stop of PCI-8164. For example: EL, ALM ..etc, these kind of INT can not be masked. The following is the definition of these two int_status:

event_int_status : can be masked by function call ***_8164_int_*****factor()**	
Bit	Description
0	Normal Stop
1	Next command continued
2	Continuous pre-register is empty and allow users to fill new command
3	(Reserved)
4	Acceleration Start
5	Acceleration End
6	Deceleration Start
7	Deceleration End
8	(Reserved)
9	(Reserved)
10	Step-losing occur
11	General Comparator compared
12	Compared triggered for axis 0,1
13	(Reserved)
14	Latched for axis2,3
15	ORG on
16	SD on
17~31	Reserved

error_int_status : can not be masked if interrupt service is activated.

Bit	Description
0	+SL Stop
1	-SL Stop
2	Reserved
3	General Comparator Stop
4	Reserved
5	+EL
6	-EL
7	ALM
8	Reserved
9	Reserved
10	SD on then stop
11~31	Reserved

Use Thread to deal with Interrupt under Windows NT/95

In order to detect the interrupt signal from PCI-8164 under Windows NT/95, user must create a thread routine first. Then use APIs provided by PCI-8164 to get the interrupt status. The sample program is as following:

```
// Steps:
// 1. Define a Global Value to deal with interrupt event
        HANDLE hEvent[4];
        int ThreadOn;

// 2. In Initializing  Section ( you must  Initialize PCI-8164 properly first),
        I16 TotalCard
        _8164_initial(&TotalCard);
        if( TotalCard == 0 ) return 0;

// 3. Enable interrupt service and setup int factor (to mask int).
        _8164_int_enable(0,hEvent);
        _8164_set_int_factor(0,0x201);
        _8164_int_control(0,1);


// 4. Create and execute thread
        AfxBeginThread(IntThread,NULL);
```

```
// =============Thread Body ===============
        UINT IntThread(LPVOID pParam)
        {
                    U32 STS;
                    ThreadOn=1;
                     while( ThreadOn==1 )   {

                    // Wait Axis 0 Interrupt Event

                     STS=::WaitForSingleObject(hEvent[0],INFINITE);
                     if( STS==WAIT_OBJECT_0 )              {
                      _8164_get_int_status(0, &error, &event);

                    // Insert User' s code here

                                :
                                :
                                }
                    // Before exiting the program, remember to let the
                    //  thread go to end naturally by setting ThreadOn=0
                    // and set Event On.

                                ::ResetEvent(hEvent[0]);  // only under
                                                          // win95/98
                    }
                    ThreadOn=2;
                    return 0;
        }
```

PCI-8164 Interrupt Service Routine (ISR) with DOS

A DOS function library is equipped with PCI-8164 for users to develop applications under DOS environment. This library also provides some functions for users to work with ISR. It is highly recommended to write programs according to the following example for applications should work with ISR. Since PCI-bus has the ability to do IRQ sharing when multiple PCI-8164 are applied, each PCI-8164 should have a corresponding ISR. For users who use the library we provide, the names of ISR are fixed, such as: *_8164_isr0(void), _8164_isr1(void)…etc.* The sample program are described as below. It is assumed that two PCI-8164 are plugged on the slot , axis 1 and axis5 are asked to work with ISR.:

```
// header file declare
#include        " pci_8164.h"
```

```
void main(void) {
        I16 TotalCard,i;                        // Initialize  cards
        _8164_initial(&TotalCard);
        if( TotalCard == 0 ) exit(1);

        _8164_set_int_factor(0,0x1); // Set int factor
        _8164_int_control(0,1);         // enable int service

        :
        :                                         // Insert User' s Code in Main
        :                                         //

        _8164_int_control(0,0);         // disable int service

        _8164_close();                        // Close PCI-8164
}

void interrupt _8164_isr0(void)  {
        U16 irq_status;                       // Declaration
        U16 int_type;
        I16 i;
        U32 i_int_status1[4],i_int_status2[4];

        disable();                              // Stop all int service
        _8164_get_irq_status(0, &irq_status);  // Check if this card' s int
        if(irq_status)  {
                for(i=0;i<4;i++) _8164_enter_isr(i);        // enter isr
                for(i=0;i<4;i++)  {
                        _8164_get_int_type(i, &int_type); // check int type
                        if( int_type & 0x1 )   {
                                _8164_get_error_int(i, &int_status1[i]);

                // Insert User' s Code in Error INT
                //
                //

                        }
                        if( int_type & 0x2 )  {
                                _8164_get_event_int(i, &int_status2[i]);

                 // Insert User' s Code in Event INT
                 //
                 //
                        }
                }
                // end of for every axis in card0
                for(i=0;i<4;i++) _8164_leave_isr(i);
        }
        else  _8164_not_my_irq(0);

    // Send EOI
        _OUTPORTB(0x20, 0x20);
        _OUTPORTB(0xA0, 0x20);
        enable();                               // allow int service
    }

    void interrupt _8164_isr1(void){}
    void interrupt _8164_isr2(void){}
    void interrupt _8164_isr3(void){}
    void interrupt _8164_isr4(void){}
    void interrupt _8164_isr5(void){}
    void interrupt _8164_isr6(void){}
    void interrupt _8164_isr7(void){}
    void interrupt _8164_isr8(void){}
    void interrupt _8164_isr9(void){}
    void interrupt _8164_isra(void){}
    void interrupt _8164_isrb(void){}
```

So with the sample, user can get the interrupt signal about each axis in the motion control system.

Relative Function:

_8164_int_control(), _8164_set_int_factor(), _8164_int_enable(), _8164_int_disable(), _8164_get_int_status(), _8164_link_interrupt(), _8164_get_int_type(), _8164_enter_isr(), _8164_leave_isr() _8164_get_event_int(), _8164_get_error_int(), _8164_get_irq_status() _8164_not_my_irq(), _8164_isr0~9, a, b

: refer to section 6.14

5

Motion Creator

After installing all the hardware properly according to Chapter 2 and 3, it is necessary to correctly configure cards and double check before running. This chapter gives guidelines for establishing a control system and manually exercising the PCI-8164 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-8164 cards.

Note that Motion Creator is available only for Windows 95/98 or Windows NT with the screen resolution higher than 800x600 environment and can not run on DOS.

5.1 Execute Motion Creator

After installing the software driver of PCI-8164 on Windows 95/98/NT/2000, the motion creator program can be find in <chosen path >/PCI-8134/Utility. To execute it, double click it or use desktop “Start” → “Program files” → “PCI-8164” → “Motion Creator”.

5.2 About Motion Creator

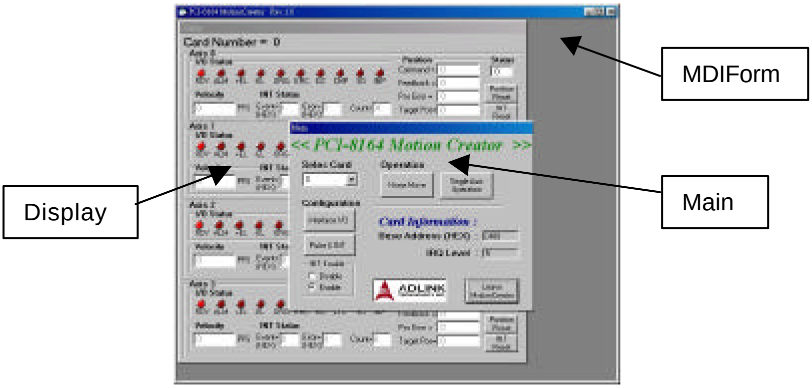
Before Running Motion Creator for PCI-8164, the following issues should be keep in mind.

1. Motion Creator is a Multiple Document Interface (MDI) program written by VB 5.0, and is available only for Windows 95/98 or Windows NT/2000 with the screen resolution higher than 800x600 environment and can not run on DOS.
2. Motion Creator allows users to save settings or configurations for PCI-8164 cards and those saved configurations will be loaded automatically when motion creator is executed later again. The two files ***8164.ini*** and ***8164MC.ini*** in ***windows root directory*** are used to save all settings and configurations.
3. To duplicate configurations from one system to another system, just copy 8164.ini and 8164MC.ini into windows root directory.
4. If users want to use the configurations set by Motion Creator, the DLL function call “ “ is helpful. After calling this function in user’ s program, user can use those PCI-8164 cards as the same configuration as set by Motion Creator.

5.3 Motion Creator Form Introducing

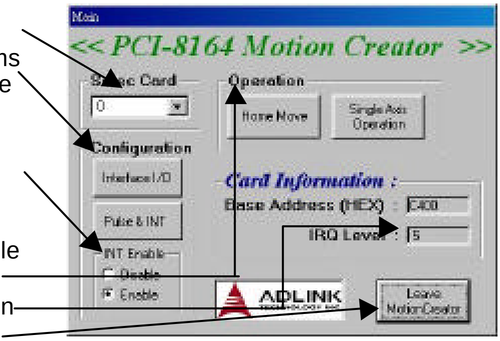
5.3.1 Initial forms:

Since the Motion Creator is MDI program, the MDIForm window will appear when executing Motion Creator. In the MDIForm you will find 2 child forms: “Main” & “Display”. The following Figure shows the initial view after running Motion Creator.

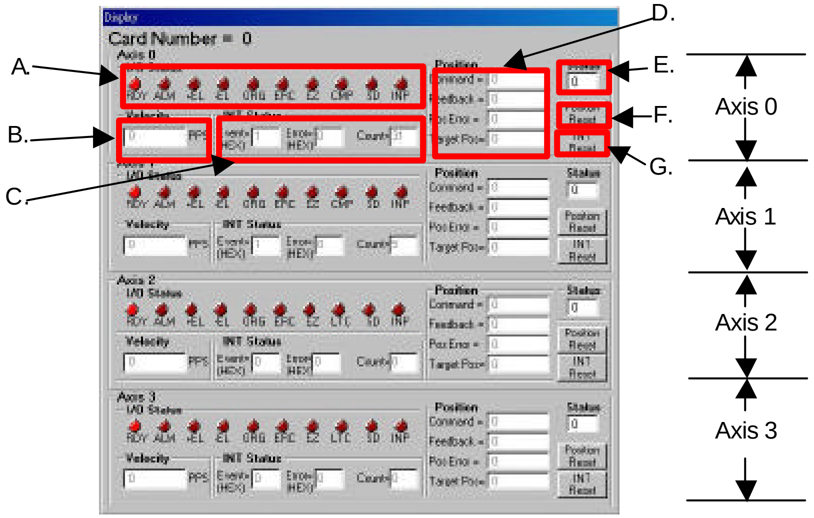


1. **MDIForm**: The MDI father window, all child forms are constrained in it.
2. **Main**: the main form is used to:

- Select operating card
- Call configuration forms (“Interface I/O”, “Pulse & INT”)
- Enable/Diable the interrupt function for individual card
- Call operation forms (“Home Move”, “Single Axis Operation”)
- Show Card Information
- Leave Motion Creator



3. **Display**: the display form is used to show information of motion I/O, position and velocity, and INT status for every of the four 4 axes in one card.



A. **I/O Status:** The status of motion I/O. Light-On means Active, while Light-Off indicates inactive. The related function is *_8164_get_io_status().*

B. **Velocity:** The absolute value of velocity in unit of PPS. The related function is *_8164_get_current_speed().*

C. INT Status:

Event: display of event_int_status in Hex value. The related function is *_8164_get_int_status().*

Error: display of error_int_status in Hex value. The related function is *_8164_get_int_status().*

Count: total count of interrupt.

D. Position:

Command: display value of command counter. The related function is *_8164_get_command().*

Feedback: display value of feedback position counter. The related function is *_8164_get_position()*

Pos Error: display value of position error counter. The related function is *_8164_get_error_counter().*

Target Pos: display value of target position recorder. The related function is *_8164_get_target_pos().*

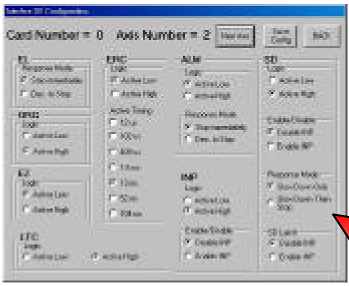
E. **Motion Status:** display return value of _8164_motion_done function. The related function is *_8164_motion_done().*

F. **Position Reset Button:** click this button will clear all position counter to ‘ 0’ . The related functions are:
_8164_set_position(),
_8164_set_command(),
_8164_reset_error_counter(),
_8164_reset_target_pos()

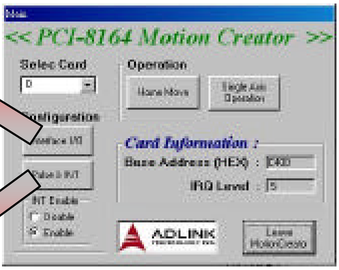
G. **INT Reset Button:** click this button will clear all INT status and INT counter to ‘ 0’ .

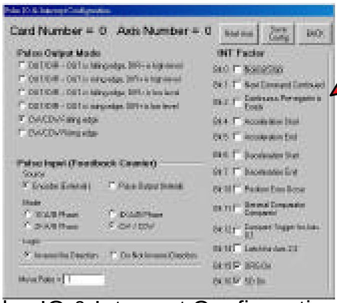
5.3.2 Configuration forms:

There are two configuration forms in Motion Creator of PCI-8164. They are “ Interface I/O Configuration” and “Pulse IO & Interrupt Configuration” . To call I/O configuration form, click the “ Interface I/O” button in “ Main” form. To call Pulse I/O and INT configuration form, click the “ Pulse & INT” button in “ Main” form.



Interface I/O Configuration Form

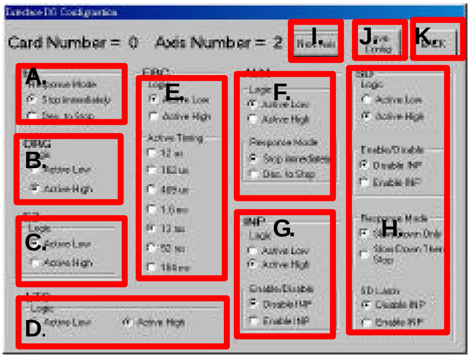




Pulse IO & Interrupt Configuration Form

Interface I/O Configuration Form

In this form user can set the configuration of EL, ORG, EZ, ERC, ALM, INP, SD, and LTC.



A **EL Response mode:** Select the response mode of EL signal. The related function call is *_8164_set_el()*.

B **ORG Logic:** Select the logic of ORG signal. The related function call is *_8164_set_home_config().*

C **EZ Logic:** Select the logic of EZ signal. The related function call is *_8164_set_home_config().*

D **LTC Logic:** Select the logic of LTC signal. The related function call is *_8164_set_ltc_logic().*

E **ERC Logic and Active timing:** Select the Logic and Active timing of ERC signal. The related function call is *_8164_set_erc()*.

F **ALM Logic and Response mode:** Select logic and response mode of ALM signal. The related function call is *_8164_set_alm().*

G **INP Logic and Enable/Disable selection:** Select logic and Enable/ Disable the INP signal. The related function call is *_8164_set_inp().*

H **SD Configuration:** Configuration of SD signal. The related function call is *_8164_set_sd().*

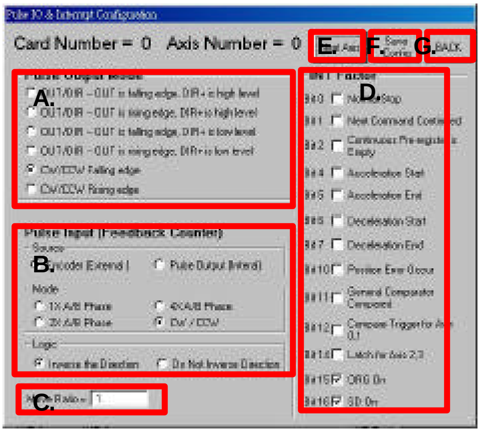
I **Next Axis:** Click this button to change operating axis.

J **Save Config:** Click this button to save current configuration to 8164.ini.

K **Back:** Click this button to go back “ main” form.

Pulse IO & Interrupt Configuration Form

In this form user can set the configuration of pulse input/output, move ration, and INT factor.


Pulse IO & Interrupt Configuration

Card Number = 0   Axis Number = 0

Pulse Output Mode
- OUT/DIR – OUT is falling edge, DIR+ is high level
- OUT/DIR – OUT is rising edge, DIR+ is high level
- OUT/DIR – OUT is falling edge, DIR+ is low level
- OUT/DIR – OUT is rising edge, DIR+ is low level
- [✓ CW/CCW Falling edge]
- CW/CCW Rising edge

Pulse Input (Feedback Counter)

Source: [✓ Encoder (External)]   Pulse Output (Internal)

Mode: 1X A/B Phase   4X A/B Phase   2X A/B Phase   [✓ CW / CCW]

Logic: [✓ Inverse the Direction]   Do Not Inverse Direction

Move Ratio = 1

INT Factor
- Bit 0 Normal Stop
- Bit 1 Next Command Continued
- Bit 2 Continuous Pre-register is Empty
- Bit 4 Acceleration Start
- Bit 5 Acceleration End
- Bit 6 Deceleration Start
- Bit 7 Deceleration End
- Bit 10 Position Error Occur
- Bit 11 General Comparator Compared
- Bit 12 Compare Trigger for Axis 0,1
- Bit 14 Latch for Axis 2,3
- [✓ Bit 15 ORG On]
- [✓ Bit 16 SD On]

A **Pulse Output Mode:** Select the output mode of pulse signal (OUT/DIR). The related function call is *_8164_set_pls_outmode().*

B **Pulse Input :** Set the configurations of Pulse input signal(EA/EB). The related function call is *_8164_set_pls_iptmode(), _8164_set_feedback_src().*

C **Move Ratio:** Set the move ratio (feedback / pulse command)for current target axis. The value should not be ‘ 0’ . The related function call is *_8164_set_move_ratio().*

D **INT Factor:** Select factors to initiate the event int. The related function call is *_8164_set_int_factor().*

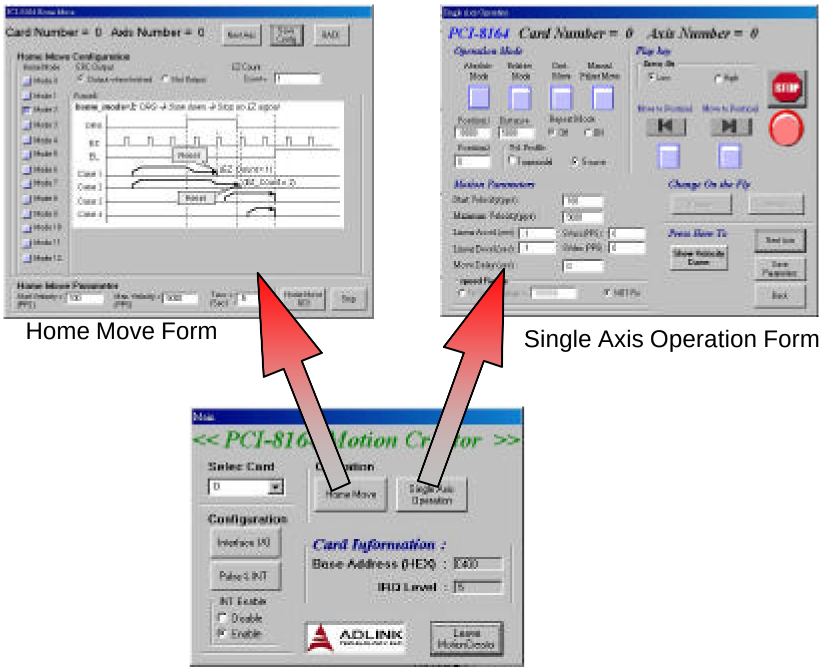
E **Next Axis:** Click this button to change operating axis.

F **Save Config:** Click this button to save current configuration to 8164.ini.

G **Back:** Click this button to go back “ main” form.

5.3.3 Operation Forms

There are two operating forms in Motion Creator of PCI-8164. They are “Home Move” and “Single Axis Operation”. To call home move form, click the “Home Move” button in “Main” form. To call Single Axis Operation form, click the “Single Axis Operation” button in “Main” form.

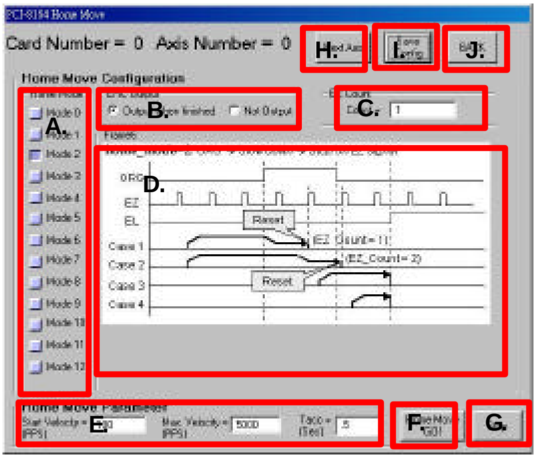


Home Move Form

Single Axis Operation Form

Home Move Form

In this form user can learn and manipulate the 13 home move functions provide by PCI-8164.



A **Home Mode:** Select the home return mode. In PCI-8164, there are 13 modes available. The related function is *_8164_set_home_config().*

B **ERC Output:** Select if the ERC signal will be sent or not when home move completes. The related function is *_8164_set_home_config().*

C **EZ Count:** Set the EZ count number, which is effective on certain home return modes. The related function is *_8164_set_home_config().*

D **Home Mode figure:** The figure showed here explains the action of individual home mode.

E **Home Move Parameters:** Set home move parameters

Start Velocity: Set the start velocity for home move in unit of PPS.

Max. Velocity: Set the maximum velocity for home move in unit of PPS.

Tacc: Set the acceleration time for home move in unit of second.

F **Home Move Go! :** Click this button to start home return motion. The related function is *_8164_home_move().*

G **Stop:** Click this button to immediately stop motion. The related function is *_8164_emg_stop().*

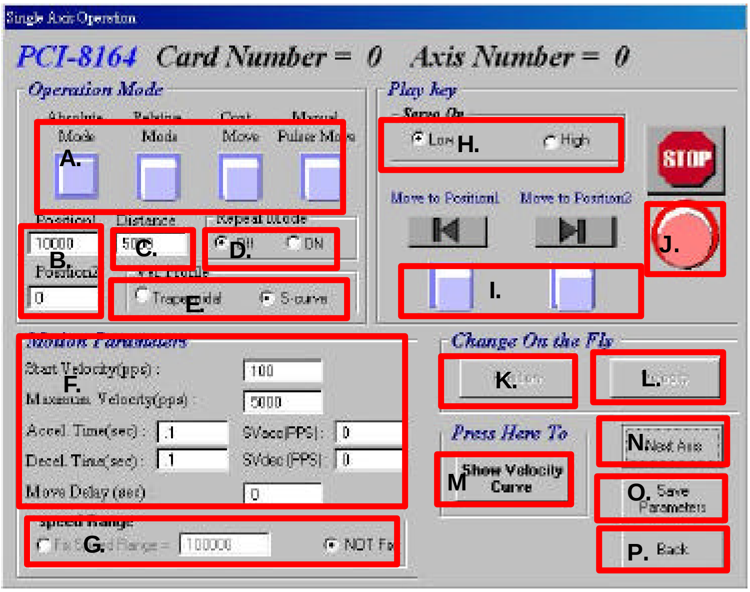
H **Next Axis:** Click this button to change operating axis.

I **Save Config:** Click this button to save current configuration to 8164.ini.

J **Back:** Click this button to go back "main" form.

Single Axis Operation

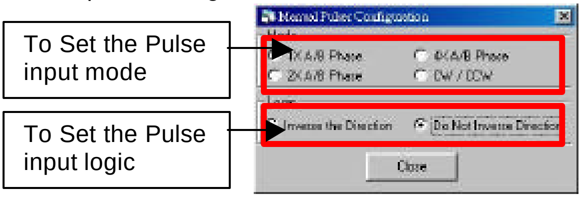
In this form user can learn and manipulate the single axis motion functions provide by PCI-8164, including velocity mode motion, preset relative/absolute motion and manual pulser move.



Position1: 10000
Position2: 0
Distance: 5000
Repeat Mode: [✓ OFF] ON
Vel. Profile: Trapezoidal [✓ S-curve]
Start Velocity(pps) : 100
Maximum Velocity(pps) : 5000
Accel. Time(sec) : .1
Decel. Time(sec) : .1
SVacc(PPS) : 0
SVdec(PPS) : 0
Move Delay (sec) 0
Speed Range: Fix Speed Range = 100000 [✓ NOT Fix]
Servo On: [✓ Low] High

A. **Operation Mode:** Select operation mode.

- Absolute Mode: "Position1" and "position2" will be used as absolution target position for motion. The related function is *_8164_start_ta_move()*, *_8164_start_sa_move()*.
- Relative Mode: "Distance will" be used as relative displacement for motion. The related function is *_8164_start_tr_move(), _8164_start_sr_move()*.
- Cont. Move: Velocity motion mode. The related function is *_8164_tv_move(), _8164_start_sv_move()*.
- Manual Pulser Move: Manual Pulser motion. Click this button will invoke the manual pulse configuration window.



B **Position:** Set the absolute position for "Absolute Mode". It is only effective when "Absolute Mode" is selected.

C **Distance:** Set the relative dstance for "Relative Mode". It is only effective when "Relative Mode" is selected.

D **Repeat Mode:** When "On" is selected, the motion will go in repeat mode(**forward←→ backward** or **position1 ←→ position2**). It is only effective when "Relative Mode" or "Absolute Mode" is selected.

E **Vel. Profile:** Select the velocity profile. Both Trapezoidal and S-Curve are available for "Absolute Mode", "Relative Mode" and "Cont. Move".

F **Motion Parameters:** Set the parameters for single axis motion. These parameter is meaningless if "Manual Pulse Move" is selected, since the velocity and moving distance is decided by pulser input.

- Start Velocity: Set the start velocity of motion in unit of PPS. In "Absolute Mode" or "Relative Mode", only the value is effective. ie, -100.0 is the same as 100.0. In "Cont. Move", both the value and sing is effective. –100.0 means 100.0 in minus direction.
- Maximum Velocity: Set the maximum velocity of motion in unit of PPS. In "Absolute Mode" or "Relative Mode", only the value is effective. ie, -5000.0 is the same as 5000.0. In "Cont. Move", both the value and sing is effective. –5000.0 means 5000.0 in minus direction.
- Accel. Time :Set the acceleration time in unit of second.
- Decel. Time :Set the deceleration time in unit of second.
- SVacc: Set the S-curve range during acceleration in unit of PPS.
- SVdec: Set the S-curve range during deceleration in unit of PPS.
- Move Delay: This setting is effective only when repeat mode is set "On". It will cause PCI-8164 to delay specified time before it continue next motion.

G **Speed Range:** Set the max speed of motion. If "Not Fix" is selected, the "Maximum Speed" will automatically become the maximum speed range, which can not be exceeded by on-the-fly velocity change.

H **Servo On:** Set the SVON signal output status. The related function is *_8164_set_servo()*.

I **Play Key:**

Left play button: Click this button will cause PCI-8164 start to outlet pulses according to previous setting.

In "*Absolute Mode*", it cause axis move to position1.
In "*Relative Mode*", it cause axis move forward.
In "*Cont. Move*", it cause axis start to move according to the velocity setting.
In "*Manual Pulser Move*", it cause axis get into pulser move. The speed limit is the value set by "Maximum Velocity"

Right play button: Click this button will cause PCI-8164 start to outlet pulses according to previous setting.

In "*Absolute Mode*", it cause axis move to position2.
In "*Relative Mode*", it cause axis move backward.
In "*Cont. Move*", it cause axis start to move according to the velocity setting, but the other direction.

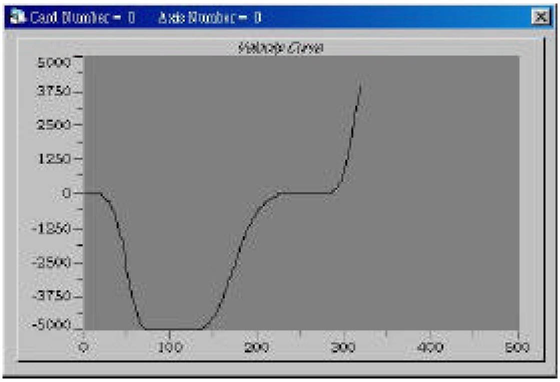
In "*Manual Pulser Move*", it cause axis get into pulser move. The speed limit is the value set by "Maximum Velocity"

J **Stop Button:** Click this button will cause PCI-8164 to decelerate to stop. The deceleration time is defined in "Decel. Time". The related function is *_8164_sd_stop().*

K **Change Position On The Fly Button:** When this button is enabled, users can change the target position of current motion. The new position must be defined in "Position2". The related function is *_8164_v_change().*

L **Change Velocity On The Fly Button:** When this button is enabled, users can change the velocity of current motion. The new velocity must be defined in "Maximum Velocity". The related function is *_8164_p_change().*

M **Show Velocity Curve Button:** Click this button will invoke a window showing velocity vs. time curve. In this curve, every 100ms a new velocity data will be added in. To close it, click this button again. To clear data, click on the curve.



N **Next Axis:** Click this button to change operating axis.
O **Save Config:** Click this button to save current configuration to 8164.ini.
P **Back:** Click this button to go back "main" form.

6

Function Library

This chapter describes the supporting software for PCI-8164 cards. User can use these functions to develop application program in C or Visual Basic or C++ language. If Delphi is used as programming environment, it is necessary to transform the header file,8164.h, manually.

6.1 List of Functions

Initialization: **Section 6.3**

Function Name	Description
_8164_initial	Software initialization
_8164_close	Software Close
_8164_get_base_addr	Get base address of PCI-8164
_8164_get_irq_channel	Get the PCI-8164 card' s IRQ number
_8164_config_from_file	Configure PCI-8164 cards according to configuration file ie. 8164.ini, which is created by Motion Creator.

Pulse Input/Output Configuration ***Section 6.4***

Function Name	Description
_8164_set_pls_outmode	Set pulse command output mode
_8164_set_pls_iptmode	Set encoder input mode
_8164_set_feedback_src	Set counter input source

Velocity mode motion *Section 6.5*

Function Name	Description
_8164_tv_move	Accelerate an axis to a constant velocity with trapezoidal profile
_8164_sv_move	Accelerate an axis to a constant velocity with S-curve profile
_8164_v_change	Change speed on the fly
_8164_sd_stop	Decelerate to stop
_8164_emg_stop	Immediately stop
_8164_fix_speed_range	Define the speed range
_8164_unfix_speed_range	Release the speed range constrain
_8164_get_current_speed	Get current speed

Single Axis Position Mode *Section 6.6*

Function Name	Description
_8164_start_tr_move	Begin a relative trapezoidal profile move
_8164_start_ta_move	Begin an absolute trapezoidal profile move
_8164_start_sr_move	Begin a relative S-curve profile move
_8164_start_sa_move	Begin an absolute S-curve profile move
_8164_set_move_ratio	Set the ratio of command pulse and feedback pulse.
_8164_p_change	Change position on the fly
_8164_set_pcs_logic	Set the logic of PCS (Position Change Signal)
_8164_set_sd_pin	Set SD/PCS pin
_8164_backlash_comp	Set backlash corrective pulse for compensation
_8164_suppress_vibration	Set vibration suppressing timing

Function Name	**Description**
_8164_start_tr_move_xy	Begin a relative 2-axis linear interpolation for X & Y, with trapezoidal profile
_8164_start_ta_move_xy	Begin a absolute 2-axis linear interpolation for X & Y, with trapezoidal profile
_8164_start_sr_move_xy	Begin a relative 2-axis linear interpolation for X & Y, with S-curve profile
_8164_start_sa_move_xy	Begin a absolute 2-axis linear interpolation for X & Y, with S-curve profile
_8164_start_tr_move_zu	Begin a relative 2-axis linear interpolation for Z & U, with trapezoidal profile
_8164_start_ta_move_zu	Begin a absolute 2-axis linear interpolation for Z & U, with trapezoidal profile
_8164_start_sr_move_zu	Begin a relative 2-axis linear interpolation for Z & U, with S-curve profile
_8164_start_sa_move_zu	Begin a absolute 2-axis linear interpolation for Z & U, with S-curve profile
_8164_start_tr_line2	Begin a relative 2-axis linear interpolation for any 2 axes, with trapezoidal profile
_8164_start_sr_line2	Begin a relative 2-axis linear interpolation for any 2 axes, with S-curve profile
_8164_start_ta_line2	Begin a absolute 2-axis linear interpolation for any 2 axes, with trapezoidal profile
_8164_start_sa_line2	Begin a absolute 2-axis linear interpolation for any 2 axes, with S-curve profile
_8164_start_tr_line3	Begin a relative 3-axis linear interpolation with trapezoidal profile
_8164_start_sr_line3	Begin a relative 3-axis linear interpolation with S-curve profile
_8164_start_ta_line3	Begin a absolute 3-axis linear interpolation with trapezoidal profile
_8164_start_sa_line3	Begin a absolute 3-axis linear interpolation with S-curve profile,
_8164_start_tr_line4	Begin a relative 4-axis linear interpolation with trapezoidal profile
_8164_start_sr_line4	Begin a relative 4-axis linear interpolation with S-curve profile
_8164_start_ta_line4	Begin a absolute 4-axis linear interpolation with trapezoidal profile
_8164_start_sa_line4	Begin a absolute 4-axis linear interpolation with S-curve profile,

Circular Interpolation Motion ***Section 6.8***

Function Name	Description
_8164_start_a_arc_xy	Begin a absolute circular interpolation for X & Y
_8164_start_r_arc_xy	Begin a relative circular interpolation for X & Y
_8164_start_a_arc_zu	Begin a absolute circular interpolation for Z & U
_8164_start_r_arc_zu	Begin a relative circular interpolation for Z & U
_8164_start_a_arc2	Begin a absolute circular interpolation for any 2 of the 4 axes
_8164_start_r_arc2	Begin a relative circular interpolation for any 2 of the 4 axes

Home Return Mode ***Section 6.9***

Function Name	Description
_8164_set_home_config	Set the home/index logic configuration
_8164_home_move	Begin a home return action

Manual Pulser Motion ***Section 6.10***

Function Name	Description
_8164_set_pulser_iptmode	Set pulser input mode
_8164_pulser_vmove	Start pulser v move
_8164_pulser_pmove	Start pulser p move
_8164_pulser_home_move	Start pulser home move

Motion Status ***Section 6.11***

Function Name	Description
_8164_motion_done	Return the motion status

Motion Interface I/O ***Section 6.12***

Function Name	Description
_8164_set_alm	Set alarm logic and operating mode
_8164_set_inp	Set INP logic and operating mode
_8164_set_erc	Set ERC logic and timing
_8164_set_servo	Set state of general purpose output pin
_8164_set_sd	Set SD logic and operating mode
_8164_set_el	Set EL logic and operating mode

Motion I/O Monitoring ***Section 6.13***

Function Name	Description
_8164_get_io_status	Get all the motion I/O status of PCI-8164

Interrupt Control ***Section 6.14***

Function Name	Description
_8164_int_control	Enable/Disable INT service
_8164_int_enable	Enable event (For Window only)
_8164_int_disable	Disable event (For Window only)
_8164_get_int_status	Get INT Status (For Window only)
_8164_link_interrupt	Set link to interrupt call back function (For Window only)
_8164_set_int_factor	Set INT factor
_8164_get_int_type	Get INT type (For DOS only)
_8164_enter_isr	Enter interrupt service routine (For DOS only)
_8164_leave_isr	Leave interrupt service routine (For DOS only)
_8164_get_event_int	Get event status (For DOS only)
_8164_get_error_int	Get error status (For DOS only)
_8164_get_irq_status	Get IRQ status (For DOS only)
_8164_not_my_irq	Not My IRQ (For DOS only)
_8164_isr0~9, a, b	Interrupt service routine (For DOS only)

Position Control and Counters ***Section 6.15***

Function Name	Description
_8164_get_position	Get the value of feedback position counter
_8164_set_position	Set the feedback position counter
_8164_get_command	Get the value of command position counter
_8164_set_command	Set the command position counter
_8164_get_error_counter	Get the value of position error counter
_8164_reset_error_counter	Reset the position error counter
_8164_get_general_counter	Get the value of general counter
_8164_set_general_counter	Set the general counter
_8164_get_target_pos	Get the value of target position recorder
_8164_reset_target_pos	Reset target position recorder
_8164_get_rest_command	Get remain pulse till end of motion
_8164_check_rdp	Check the ramping down point data

Position Compare and Latch ***Section 6.16***

Function Name	**Description**
_8164_set_ltc_logic	Set the LTC logic
_8164_get_latch_data	Get latched counter data
_8164_set_soft_limit	Set soft limit
_8164_enable_soft_limit	Enable soft limit function
_8164_disable_soft_limit	Disable soft limit function
_8164_set_error_counter_check	Step-losing detection
_8164_set_general_comparator	Set general-purposed comparator
_8164_set_trigger_comparator	Set Trigger comparator
_8164_set_trigger_type	Set the trigger output type
_8164_check_compare_data	Check current comparator data
_8164_check_compare_status	Check current comparator status
_8164_set_auto_compare	Set comparing data source for auto loading
_8164_build_compare_function	Build compare data via constant interval
_8164_build_compare_table	Build compare data via compare table
_8164_cmp_v_change	Speed change by comparator

Continuous motion ***Section 6.17***

Function Name	**Description**
_8164_set_continuous_move	Enable continuous motion for absolute motion
_8164_check_continuous_buffer	Check if the buffer is empty

General-purposed TTL Output ***Section 6.18***

Function Name	**Description**
_8164_d_output	Digital Output
_8164_get_dio_status	Get DO status

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-8164.H**. These data types are used by PCI-8164 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

The functions of PCI-8164' s software drivers use full-names to represent the functions' real meaning. The naming convention rules are :

In ' C' programming Environment :

_{hardware_model}_{action_name}. e.g. **`_8164_Initial()`**.

In order to recognize the difference between C library and VB library, a capital "B" is put on the head of each function name e.g. **`B_8164_Initial()`**.

6.3 Initialization

@ Name

_8164_Initial – Software Initialization for PCI-8164
_8164_Close – Software release resources of PCI-8164
_8164_get_base_addr – Get the base address of PCI-8164
_8164_get_irq_channel – Get the PCI-8164 card' s IRQ number
_8164_config_from_file – Configure PCI-8164 card according to configuration file ie. 8164.ini.

@ Description

_8164_Initial :

This function is used to initialize PCI-8164 card. All PCI-8164 cards must be initialized by this function before calling other functions.

_8164_Close :

This function is used to close PCI-8164 card and release the PCI-8164 related resources, which should be called at the end of an application.

_8164_get_irq_channel :

This function is used to get the PCI-8164 card' s IRQ number.

_8164_get_base_addr:

This function is used to get the PCI-8164 card' s base address.

_8164_config_from_file:

This function is used to load the configuration of PCI-8164 according to specified file. By using **Motion Creator**, user can test and configure PCI-8164 correctly. After pressing "save config" button, the 8164.ini file in window directory is used to record the configurations. By specifying it in the parameter, the configuration will be automatically loaded.

When this function is executed, all PCI-8164 cards in the system will be configured as the following functions were called according to parameters recorded in 8164.ini.

_8164_set_pls_outmode
_8164_set_feedback_src
_8164_set_pls_iptmode
_8164_set_home_config
_8164_set_int_factor
_8164_set_el
_8164_set_ltc_logic
_8164_set_erc
_8164_set_sd
_8164_set_alm
_8164_set_inp
_8164_set_move_ratio

@ Syntax

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

```
I16 _8164_initial(I16 *existCards);
I16 _8164_close(void);
I16 _8164_get_irq_channel(I16 cardNo, U16 *irq_no );
I16 _8164_get_base_addr(I16 cardNo, U16 *base_addr );
I16 _8164_config_from_file(char *filename);
```

Visual Basic (Windows 95/NT)

```
B_8164_initial (existCards As Integer) As Integer
B_8164_close () As Integer
B_8164_get_irq_channel (ByVal CardNo As Integer, irq_no As Integer) As
        Integer
B_8164_get_base_addr (ByVal CardNo As Integer, base_addr As Integer)
        As Integer
B_8164_config_from_file(ByVal filename As string)as integer
```

@ Argument

***existCards**: numbers of existing PCI-8164 cards
cardNo: The PCI-8164 card index number.
***irq_no**: Irq number of specified PCI-8164 card.
***base_addr**: base address of specified PCI-8164 card
***Filename**: The specified filename recording the configuration of PCI-8164. This file must be created by **Motion Creator** of PCI-8164.

@ Return Code

ERR_NoError
ERR_NoCardFound
ERR_PCIBiosNotExist
ERR_ConigFileOpenError

6.4 Pulse Input/Output Configuration

@ Name

_8164_set_pls_outmode – Set the configuration for pulse command output.
_8164_set_pls_iptmode – Set the configuration for feedback pulse input.
_8164_set_feedback_src – Enable/Disable the external feedback pulse input

@ Description

_8164_set_pls_outmode:

Configure the output modes of command pulse. There are 6 modes for command pulse output.

_8164_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 **Src** parameter in **_8164_set_feedback_src ()** function is enabled.

_8164_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 ***_8164*_set_pls_iptmode()** function. Or the counter will count the command pulse output.

@ Syntax

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

```
I16 _8164_set_pls_outmode(I16 AxisNo, I16 pls_outmode);
I16 _8164_set_pls_iptmode(I16 AxisNo, I16 pls_iptmode, I16 pls_logic);
I16 _8164_set_feedback_src(I16 AxisNo, I16 Src);
```

Visual Basic (Windows 95/NT)

```
B_8164_set_pls_outmode (ByVal AxisNo As Integer, ByVal pls_outmode
        As Integer) As Integer
B_8164_set_pls_iptmode (ByVal AxisNo As Integer, ByVal pls_iptmode As
        Integer, ByVal pls_logic As Integer) As Integer
B_8164_set_feedback_src (ByVal AxisNo As Integer, ByVal Src As Integer)
        As Integer
```

@ Argument

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

pls_outmode: setting of command pulse output mode

Value	Meaning	
0	OUT/DIR	OUT Falling edge, DIR+ is high level
1	OUT/DIR	OUT Rising edge, DIR+ is high level
2	OUT/DIR	OUT Falling edge, DIR+ is low level
3	OUT/DIR	OUT Rising edge, DIR+ is high level
4	CW/CCW	Falling edge
5	CW/CCW	Rising edge

pls_iptmode: setting of encoder feedback pulse input mode

Value	Meaning
0	1X A/B
1	2X A/B

2	4X A/B
3	CW/CCW

pls_logic: Logic of encoder feedback pulse

pls_logic=0, Normal low.
pls_logic=1, Normal high

Src: Counter source

Value	Meaning
0	External Feedback
1	Command pulse

@ Return Code

ERR_NoError

6.5 Velocity mode motion

@ Name

_8164_tv_move – Accelerate an axis to a constant velocity with trapezoidal profile
_8164_sv_move – Accelerate an axis to a constant velocity with S-curve profile
_8164_v_change –Change speed on the fly
_8164_sd_stop –Decelerate to stop
_8164_emg_stop –Immediately stop
_8164_fix_speed_range – Define the speed range
_8164_unfix_speed_range – Release the speed range constrain
_8164_get_current_speed – Get current speed

@ Description

_8164_tv_move:

This function is to accelerate an axis to the specified constant velocity with trapezoidal profile. 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.

_8164_sv_move:

This function is to accelerate an axis to the specified constant velocity with S-curve profile. 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.

_8164_v_change:

This function changes the moving velocity with trapezoidal profile or S-curve profile. Before calling this function, it is necessary to define the speed range by _8164_fix_speed_range. _8164_v_change is also applicable on pre-set motion. Note: The velocity profile is decided by original motion profile. When using in S-curve, please set the motion to be pure S-curve. There are some limitations for this function, please refer to section 4.6.1 before use it.

_8164_sd_stop:

This function is used to decelerate an axis to stop with trapezoidal profile or S-curve profile. This function is also useful when ***preset move*** (both trapezoidal and S-curve motion), ***manual move*** or ***home return*** function is performed. Note: The velocity profile is decided by original motion profile.

_8164_emg_stop:

This function is used to immediately stop an axis. This function is also useful when ***preset move*** (both trapezoidal and S-curve motion), ***manual move*** or ***home return*** function is performed.

_8164_fix_speed_range

This function is used to define the speed range. It should be called before starting motion that may contains velocity changing.

_8164_unfix_speed_range

This function is used to Release the speed range constrain.

_8164_get_current_speed

This function is used to read current pulse output rate of specified axis. It is applicable in any time and any operating mode.

@ Syntax

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

```
I16 _8164_tv_move(I16 AxisNo, F64 StrVel, F64 MaxVel, F64 Tacc);
I16 _8164_sv_move(I16 AxisNo, F64 StrVel, F64 MaxVel, F64 Tacc, F64
        SVacc);
I16 _8164_v_change(I16 AxisNo, F64 NewVel, F64 Tacc);
I16 _8164_sd_stop(I16 AxisNo,F64 Tdec);
I16 _8164_emg_stop(I16 AxisNo);
F64 _8164_fix_speed_range(I16 AxisNo, F64 MaxVel);
I16 _8164_unfix_speed_range(I16 AxisNo);
I16 _8164_get_current_speed(I16 AxisNo, F64 *speed);
```

Visual Basic (Windows 95/NT)

```
B_8164_tv_move (ByVal AxisNo As Integer, ByVal StrVel As Double, ByVal
        MaxVel As Double, ByVal Tacc As Double) As Integer
B_8164_sv_move (ByVal AxisNo As Integer, ByVal StrVel As Double,
        ByVal MaxVel As Double, ByVal Tacc As Double, ByVal SVacc As
        Double) As Integer
B_8164_v_change (ByVal AxisNo As Integer, ByVal NewVel As Double,
        ByVal TimeSecond As Double) As Integer
B_8164_sd_stop (ByVal AxisNo As Integer, ByVal Tdec As Double) As
        Integer
B_8164_emg_stop (ByVal AxisNo As Integer) As Integer
B_8164_fix_speed_range (ByVal AxisNo As Integer, ByVal MaxVel As
        Double) As Integer
B_8164_unfix_speed_range (ByVal AxisNo As Integer) As Integer
B_8164_get_current_speed (ByVal AxisNo As Integer, Speed As Double)
        As Integer
```

@ Argument

AxisNo: axis number designated to move or stop.
StrVel: starting velocity in unit of pulse per second
MaxVel: maximum velocity in unit of pulse per second
Tacc: specified acceleration time in unit of second
SVacc: specified velocity interval in which S-curve acceleration is performed.
Note: SVacc = 0, for pure S-Curve
NewVel: New velocity in unit of pulse per second
Tdec: specified deceleration time in unit of second
***Speed:** Variable to save current speed.
(speed range: 0~6553500)

@ Return Code

ERR_NoError
ERR_SpeedError
ERR_SpeedChangeError
ERR_SlowDownPointError
ERR_AxisAlreadyStop

6.6 Single Axis Position Mode

@ Name

_8164_start_tr_move – Begin a relative trapezoidal profile move
_8164_start_ta_move – Begin an absolute trapezoidal profile move
_8164_start_sr_move – Begin a relative S-curve profile move
_8164_start_sa_move – Begin an absolute S-curve profile move
_8164_set_move_ratio –Set the ratio of command pulse and feedback pulse.
_8164_p_change – Change position on the fly
_8164_set_pcs_logic –Set the logic of PCS (Position Change Signal) pin
_8164_set_sd_pin –Set SD/PCS pin
_8164_backlash_comp – Set backlash compensating pulse for compensation
_8164_suppress_vibration – Set vibration suppressing timing

@ Description

General: *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 automatically lower the MaxVel ,and the Tacc, Tdec, VSacc, VSdec, will also become shorter while the dV/dt(acceleration / deceleration) and d(dV/dt)/dt (jerk) keep unchanged.*

_8164_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 with trapezoidal profile. The acceleration and deceleration time is specified independently. It won' t let the program wait for motion completion but immediately return control to the program.

_8164_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 with trapezoidal profile. The acceleration and deceleration time is specified independently. It won' t let the program wait for motion completion but immediately return control to the program.

_8164_start_sr_move:
This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the relative distance with S-curve profile. The acceleration and deceleration time is specified independently. It won' t let the program wait for motion completion but immediately return control to the program.

_8164_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 with S-curve profile. The acceleration and deceleration time is specified independently. It won' t let the program

wait for motion completion but immediately return control to the program.

_8164_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***.

_8164_p_change

This function is used to change target position on the fly. There are some limitations for this function. Please refer to section 4.6.2 before use it.

_8164_set_pcs_logic :

This function is used to set the logic of Position Change Signal (pcs). The PCS share the same pin with SD signal. Only when the SD/PCS pin was set to PCS by _8164_set_sd_pin, this _8164_set_pcs_logic function becomes effective.

_8164_set_sd_pin :

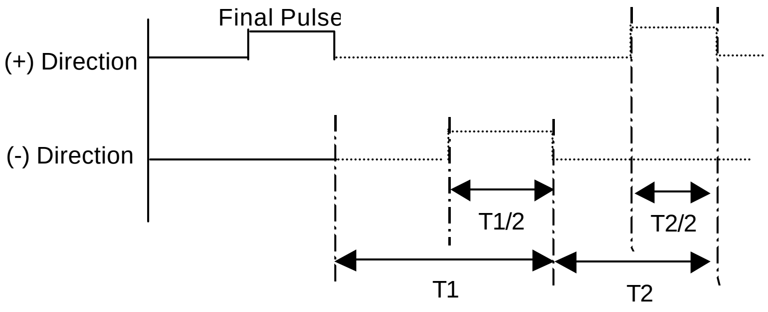
This function is used to set the operating mode of SD pin. The SD pin may be used either as Slow-Down signal input or as Position Change Signal (PCS) input. Please refer to section 4.3.1

_8164_backlash_comp :

Whenever direction change is occurred, The PCI-8164 outputs backlash corrective pulses before sending commands. This function is used set the compensation pulse numbers.

_8164_suppress_vibration

This function is used to suppress vibration of mechanical system by outputting single pulse for negative direction and then single pulse for positive direction right after completion of command movement.



@ Syntax

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

```
I16 _8164_start_tr_move(I16 AxisNo, F64 Dist, F64 StrVel, F64 MaxVel,
        F64 Tacc,F64 Tdec);
I16 _8164_start_ta_move(I16 AxisNo, F64 Pos, F64 StrVel, F64 MaxVel,
        F64 Tacc, F64 Tdec);
I16 _8164_start_sr_move(I16 AxisNo, F64 Dist, F64 StrVel, F64 MaxVel,
        F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
```

I16 _8164_start_sa_move(I16 AxisNo, F64 Pos, F64 StrVel, F64 MaxVel,
    F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8164_set_move_ratio(I16 AxisNo, F64 move_ratio);
I16 _8164_p_change(I16 AxisNo, F64 NewPos);
I16 _8164_set_pcs_logic(I16 AxisNo, I16 pcs_logic);
I16 _8164_set_sd_pin(I16 AxisNo, I16 Type);
I16 _8164_backlash_comp(I16 AxisNo, I16 BCompPulse);
I16 _8164_suppress_vibration(I16 AxisNo, U16 T1, U16 T2);

Visual Basic (Windows 95/NT)

B_8164_start_tr_move (ByVal AxisNo As Integer, ByVal Dist As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer
B_8164_start_ta_move (ByVal AxisNo As Integer, ByVal Pos As Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal Tdec As Double) As Integer
B_8164_start_sr_move (ByVal AxisNo As Integer, 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_8164_start_sa_move (ByVal AxisNo As Integer, 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_8164_set_move_ratio (ByVal AxisNo As Integer, ByVal move_ratio As Double) As Integer
B_8164_p_change (ByVal AxisNo As Integer, ByVal NewPos As Double) As Integer
B_8164_set_pcs_logic (ByVal AxisNo As Integer, ByVal pcs_logic As Integer) As Integer
B_8164_set_sd_pin (ByVal AxisNo As Integer, ByVal Type As Integer) As Integer
B_8164_backlash_comp (ByVal AxisNo As Integer, ByVal BCompPulse As Integer, ByVal ForwardTime As Integer) As Integer
B_8164_suppress_vibration (ByVal AxisNo As Integer, ByVal ReserveTime As Integer, ByVal ForwardTime As Integer) As Integer

@ Argument

AxisNo: axis number designated to move or change position.
Dist: specified relative distance to move
Pos: specified absolute position to move
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 velocity interval in which S-curve acceleration is performed.
Note: SVacc = 0, for pure S-Curve
SVdec: specified velocity interval in which S-curve deceleration is performed.
Note: SVdec = 0, for pure S-Curve
Move_ratio: ratio of (feedback resolution)/(command resolution) , should not be 0

NewPos: specified new absolute position to move
BcompPulse: Specified number of corrective pulses
T1: Specified Reverse Time
T2: Specified Forward Time

@ Return Code

ERR_NoError
ERR_SpeedError
ERR_PChangeSlowDownPointError
ERR_MoveRatioError

6.7 Linear Interpolated Motion

@ Name

_8164_start_tr_move_xy – Begin a relative 2-axis linear interpolation for X & Y, with trapezoidal profile,
_8164_start_ta_move_xy – Begin a absolute 2-axis linear interpolation for X & Y, with trapezoidal profile,
_8164_start_sr_move_xy – Begin a relative 2-axis linear interpolation for X&Y, with S-curve profile,
_8164_start_sa_move_xy – Begin a absolute 2-axis linear interpolation for X & Y, with S-curve profile,
_8164_start_tr_move_zu – Begin a relative 2-axis linear interpolation for Z & U, with trapezoidal profile,
_8164_start_ta_move_zu – Begin a absolute 2-axis linear interpolation for Z & U, with trapezoidal profile,
_8164_start_sr_move_zu – Begin a relative 2-axis linear interpolation for Z & U, with S-curve profile,
_8164_start_sa_move_zu – Begin a absolute 2-axis linear interpolation for Z & U, with S-curve profile,
_8164_start_tr_line2 – Begin a relative 2-axis linear interpolation for any 2 axes, with trapezoidal profile,
_8164_start_sr_line2 – Begin a relative 2-axis linear interpolation for any 2 axes,, with S-curve profile
_8164_start_ta_line2 – Begin a absolute 2-axis linear interpolation for any 2 axes,, with trapezoidal profile
_8164_start_sa_line2 – Begin a absolute 2-axis linear interpolation for any 2 axes,, with S-curve profile,
_8164_start_tr_line3 – Begin a relative 3-axis linear interpolation with trapezoidal profile,
_8164_start_sr_line3 – Begin a relative 3-axis linear interpolation with S-curve profile
_8164_start_ta_line3 – Begin a absolute 3-axis linear interpolation with trapezoidal profile
_8164_start_sa_line3 – Begin a absolute 3-axis linear interpolation with S-curve profile,
_8164_start_tr_line4 – Begin a relative 4-axis linear interpolation with trapezoidal profile,
_8164_start_sr_line4 – Begin a relative 4-axis linear interpolation with S-curve profile
_8164_start_ta_line4 – Begin a absolute 4-axis linear interpolation with trapezoidal profile
_8164_start_sa_line4 – Begin a absolute 4-axis linear interpolation with S-curve profile,

@ Description

Functions	No. of interpolating axes	Velocity Profile	Relative / Absolute	Target Axes
_8164_start_tr_move_xy	2	T	R	Axis 0 & 1
_8164_start_ta_move_xy	2	T	A	Axis 0 & 1
_8164_start_sr_move_xy	2	S	R	Axis 0 & 1
_8164_start_sa_move_xy	2	S	A	Axis 0 & 1
_8164_start_tr_move_zu	2	T	R	Axis 2 & 3
_8164_start_ta_move_zu	2	T	A	Axis 2 & 3
_8164_start_sr_move_zu	2	S	R	Axis 2 & 3
_8164_start_sa_move_zu	2	S	A	Axis 2 & 3
_8164_start_tr_move_line2	2	T	R	Any 2 of 4
_8164_start_ta_move_ line2	2	T	A	Any 2 of 4
_8164_start_sr_move_ line2	2	S	R	Any 2 of 4
_8164_start_sa_move_ line2	2	S	A	Any 2 of 4
_8164_start_tr_move_ line3	3	T	R	Any 3 of 4
_8164_start_ta_move_ line3	3	T	A	Any 3 of 4
_8164_start_sr_move_ line3	3	S	R	Any 3 of 4
_8164_start_sa_move_ line3	3	S	A	Any 3 of 4
_8164_start_tr_move_ line4	4	T	R	Any 4 of 4
_8164_start_ta_move_ line4	4	T	A	Any 4 of 4
_8164_start_sr_move_ line4	4	S	R	Any 4 of 4
_8164_start_sa_move_ line4	4	S	A	Any 4 of 4

@ Syntax

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

```
I16 _8164_start_tr_move_xy(I16 CardNo, F64 DistX, F64 DistY, F64 StrVel,
        F64 MaxVel, F64 Tacc, F64 Tdec);
I16 _8164_start_ta_move_xy(I16 CardNo, F64 PosX, F64 PosY, F64 StrVel,
        F64 MaxVel, F64 Tacc, F64 Tdec);
I16 _8164_start_sr_move_xy(I16 CardNo, F64 DistX, F64 DistY, F64 StrVel,
        F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8164_start_sa_move_xy(I16 CardNo, F64 PosX, F64 PosY, F64
        StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8164_start_tr_move_zu(I16 CardNo, F64 DistX, F64 DistY, F64 StrVel,
        F64 MaxVel, F64 Tacc, F64 Tdec);
I16 _8164_start_ta_move_zu(I16 CardNo, F64 PosX, F64 PosY, F64 StrVel,
        F64 MaxVel, F64 Tacc, F64 Tdec);
I16 _8164_start_sr_move_zu(I16 CardNo, F64 DistX, F64 DistY, F64 StrVel,
        F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8164_start_sa_move_zu(I16 CardNo, F64 PosX, F64 PosY, F64
        StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8164_start_tr_line2(I16 CardNo, I16 *AxisArray, F64 DistX, F64 DistY,
```

F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);
I16 _8164_start_ta_line2(I16 CardNo, I16 *AxisArray, F64 PosX, F64 PosY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);
I16 _8164_start_sr_line2(I16 CardNo, I16 *AxisArray, F64 DistX, F64 DistY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8164_start_sa_line2(I16 CardNo, I16 *AxisArray, F64 PosX, F64 PosY, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8164_start_tr_line3(I16 CardNo, I16 *AxisArray, F64 DistX, F64 DistY, F64 DistZ, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);
I16 _8164_start_ta_line3(I16 CardNo, I16 *AxisArray, F64 PosX, F64 PosY, F64 PosZ, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);
I16 _8164_start_sr_line3(I16 CardNo, I16 *AxisArray, F64 DistX, F64 DistY, F64 DistZ, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8164_start_sa_line3(I16 CardNo, I16 *AxisArray, F64 PosX, F64 PosY, F64 PosZ, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8164_start_tr_line4(I16 CardNo, F64 DistX, F64 DistY, F64 DistZ, F64 DistU, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);
I16 _8164_start_ta_line4(I16 CardNo, F64 PosX, F64 PosY, F64 PosZ, F64 PosU, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec);
I16 _8164_start_sr_line4(I16 CardNo, F64 DistX, F64 DistY, F64 DistZ, F64 DistU, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);
I16 _8164_start_sa_line4(I16 CardNo, F64 PosX, F64 PosY, F64 PosZ, F64 PosU, F64 StrVel, F64 MaxVel, F64 Tacc, F64 Tdec, F64 SVacc, F64 SVdec);

Visual Basic (Windows 95/NT)

B_8164_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_8164_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_8164_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_8164_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_8164_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_8164_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_8164_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_8164_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
B_8164_start_tr_line2 (ByVal CardNo As Integer, AxisArray As Integer,
        ByVal DistX As Double, ByVal DistY As Double, ByVal StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal
        Tdec As Double) As Integer
B_8164_start_ta_line2 (ByVal CardNo As Integer, AxisArray As Integer,
        ByVal PosX As Double, ByVal PosY As Double, ByVal StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc As Double, ByVal
        Tdec As Double) As Integer
B_8164_start_sr_line2 (ByVal CardNo As Integer, AxisArray As Integer,
        ByVal DistX As Double, ByVal DistY 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_8164_start_sa_line2 (ByVal CardNo As Integer, AxisArray As Integer,
        ByVal PosX As Double, ByVal PosY 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_8164_start_tr_line3 (ByVal CardNo As Integer, AxisArray As Integer,
        ByVal DistX As Double, ByVal DistY As Double, ByVal DistZ As
        Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal
        Tacc As Double, ByVal Tdec As Double) As Integer
B_8164_start_ta_line3 (ByVal CardNo As Integer, AxisArray As Integer,
        ByVal PosX As Double, ByVal PosY As Double, ByVal PosZ As
        Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal
        Tacc As Double, ByVal Tdec As Double) As Integer
B_8164_start_sr_line3 (ByVal CardNo As Integer, AxisArray As Integer,
        ByVal DistX As Double, ByVal DistY As Double, ByVal DistZ 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_8164_start_sa_line3 (ByVal CardNo As Integer, AxisArray As Integer,
        ByVal PosX As Double, ByVal PosY As Double, ByVal PosZ 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_8164_start_tr_line4 (ByVal CardNo As Integer, ByVal DistX As Double,
        ByVal DistY As Double, ByVal DistZ As Double, ByVal DistU As
        Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal
        Tacc As Double, ByVal Tdec As Double) As Integer
B_8164_start_ta_line4 (ByVal CardNo As Integer, ByVal PosX As Double,
        ByVal PosY As Double, ByVal PosZ As Double, ByVal PosU As
        Double, ByVal StrVel As Double, ByVal MaxVel As Double, ByVal
```

Tacc As Double, ByVal Tdec As Double) As Integer
B_8164_start_sr_line4 (ByVal CardNo As Integer, ByVal DistX As Double, ByVal DistY As Double, ByVal DistZ As Double, ByVal DistU 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_8164_start_sa_line4 (ByVal CardNo As Integer, ByVal PosX As Double, ByVal PosY As Double, ByVal PosZ As Double, ByVal PosU 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 linear interpolation
DistX: specified relative distance of axis 0 to move
DistY: specified relative distance of axis 1 to move
DistZ: specified relative distance of axis 2 to move
DistU: specified relative distance of axis 3 to move
PosX: specified absolute position of axis 0 to move
PosY: specified absolute position of axis 1 to move
PosZ: specified absolute position of axis 2 to move
PosU: specified absolute position of axis 3 to move
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 velocity interval in which S-curve acceleration is performed.
Note: SVacc = 0, for pure S-Curve
SVdec: specified velocity interval in which S-curve deceleration is performed.
Note: SVdec = 0, for pure S-Curve
AxisArray: Array of axis number to perform interpolation.
Example: Int AxisArray[2] = {0,2}; // axis 0 & 2
Int AxisArray[3] = {0,1,3}; // axis 0,1,3
Note: AxisArray[n] must be smaller than AxisArray[m], if n<m.

@ Return Code

ERR_NoError
ERR_SpeedError
ERR_AxisArrayErrot

6.8 Circular Interpolation Motion

@ Name

_8164_start_r_arc_xy – Begin a relative circular interpolation for X & Y
_8164_start_a_arc_xy – Begin a absolute circular interpolation for X & Y
_8164_start_r_arc_zu –Begin a relative circular interpolation for Z & U
_8164_start_a_arc_zu –Begin a absolute circular interpolation for Z & U
_8164_start_r_arc2 – Begin a relative circular interpolation for any 2 axes
_8164_start_a_arc2 – Begin a absolute circular interpolation for any 2 axes

@ Description

Functions	Relative / Absolute	Target Axes
_8164_start_r_arc_xy	R	Axis 0 & 1
_8164_start_a_arc_xy	A	Axis 0 & 1
_8164_start_r_arc_zu	R	Axis 2 & 3
_8164_start_a_arc_zu	A	Axis 2 & 3
_8164_start_r_arc2	R	Any 2 of 4
_8164_start_a_arc2	A	Any 2 of 4

@ Syntax

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

```
I16 _8164_start_r_arc_xy(I16 CardNo, F64 OffsetCx, F64 OffsetCy, F64
        OffsetEx, F64 OffsetEy, I16 DIR, F64 MaxVel);
I16 _8164_start_a_arc_xy(I16 CardNo, F64 Cx, F64 Cy, F64 Ex, F64 Ey,
        I16 DIR, F64 MaxVel);
I16 _8164_start_r_arc_zu(I16 CardNo, F64 OffsetCx, F64 OffsetCy, F64
        OffsetEx, F64 OffsetEy, I16 DIR, F64 MaxVel);
I16 _8164_start_a_arc_zu(I16 CardNo, F64 Cx, F64 Cy, F64 Ex, F64 Ey,
        I16 DIR, F64 MaxVel);
I16 _8164_start_r_arc2(I16 CardNo, I16 *AxisArray, F64 OffsetCx, F64
        OffsetCy, F64 OffsetEx, F64 OffsetEy, I16 DIR, F64 MaxVel);
I16 _8164_start_a_arc2(I16 CardNo, I16 *AxisArray, F64 Cx, F64 Cy, F64
        Ex, F64 Ey, I16 DIR, F64 MaxVel);
```

Visual Basic (Windows 95/NT)

```
B_8164_start_a_arc_xy (ByVal CardNo As Integer, ByVal Cx As Double,
        ByVal Cy As Double, ByVal Ex As Double, ByVal Ey As Double,
        ByVal DIR As Integer, ByVal MaxVel As Double) As Integer
B_8164_start_r_arc_xy (ByVal CardNo As Integer, ByVal OffsetCx As
        Double, ByVal OffsetCy As Double, ByVal OffsetEx As Double,
        ByVal OffsetEy As Double, ByVal DIR As Integer, ByVal MaxVel As
        Double) As Integer
B_8164_start_a_arc_zu (ByVal CardNo As Integer, ByVal Cx As Double,
        ByVal Cy As Double, ByVal Ex As Double, ByVal Ey As Double,
        ByVal DIR As Integer, ByVal MaxVel As Double) As Integer
B_8164_start_r_arc_zu (ByVal CardNo As Integer, ByVal OffsetCx As
        Double, ByVal OffsetCy As Double, ByVal OffsetEx As Double,
```

ByVal OffsetEy As Double, ByVal DIR As Integer, ByVal MaxVel As Double) As Integer

B_8164_start_a_arc2 (ByVal CardNo As Integer, AxisArray As Integer, ByVal Cx As Double, ByVal Cy As Double, ByVal Ex As Double, ByVal Ey As Double, ByVal DIR As Integer, ByVal MaxVel As Double) As Integer

B_8164_start_r_arc2 (ByVal CardNo As Integer, AxisArray As Integer, ByVal OffsetCx As Double, ByVal OffsetCy As Double, ByVal OffsetEx As Double, ByVal OffsetEy As Double, ByVal DIR As Integer, ByVal MaxVel As Double) As Integer

@ Argument

CardNo: Card number designated to perform linear interpolation
OffsetCx: X-axis offset to center
OffsetCy: Y-axis offset to center
OffsetEx: X-axis offset to end of arc
OffsetEy: Y-axis offset to end of arc
Cx: specified X-axis absolute position of center
Cy: specified Y-axis absolute position of axis 1 to move
Ex: specified X-axis absolute position of axis 2 to move
Ey: specified Y-axis absolute position of axis 3 to move
DIR: Specified direction of arc, CW:0 , CCW:1
MaxVel: Tangential velocity in unit of pulse per second
AxisArray: Array of axis number to perform interpolation.

```
Example: Int AxisArray[2] = {0,2}; // axis 0 & 2
         Int AxisArray[2] = {1,3}; // axis 1 & 3
```

Note: AxisArray[0] must be smaller than AxisArray[1]

@ Return Code

ERR_NoError
ERR_SpeedError
ERR_AxisArrayErrot

6.9 Home Return Mode

@ Name

_8164_set_home_config – Set the configuration for home return.
_8164_home_move – Perform a home return move.

@ Description

_8164_set_home_config:

Configure the home return mode, origin & index signal(EZ) logic, EZ count and ERC output options for home_move() function. Refer to Section 4.1.8 for the setting of home_mode control.

_8146_home_move:

This function will cause the axis to perform a home return move according to the setting of **_8164_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)

```
I16 _8164_set_home_config(I16 AxisNo, I16 home_mode, I16 org_logic,
        I16 ez_logic, I16 ez_count, I16 erc_out);
I16 _8164_home_move(I16 AxisNo, F64 StrVel, F64 MaxVel, F64 Tacc);
```

Visual Basic (Windows 95/NT)

```
B_8164_set_home_config (ByVal AxisNo 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_8164_home_move (ByVal AxisNo As Integer, ByVal StrVel As Double,
        ByVal MaxVel As Double, ByVal Tacc As Double) As Integer
```

@ Argument

AxisNo: axis number designated to configure and perform home returning
home_mode: stopping modes for home return, 0~12
(Please refer to section 4.1.8)
org_logic: Action logic configuration for ORG signal
org_logic=0, active low;
org_logic=1, active high
EZ_logic: Action logic configuration for EZ signal
EZ_logic=0, active low;
EZ_logic=1, active high.
ez_count: 0~15 (Please refer to section 4.1.8)
erc_out: Set ERC output options.
erc_out =0, no erc out;
erc_out =1, erc out when homing finish
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

@ Return Code

ERR_NoError

6.10 Manual Pulser Motion

@ Name

_8164_set_pulser_iptmode - set the input signal modes of pulser
_8164_pulser_vmove – manual pulser v_move
_8164_pulser_pmove – manual pulser p_moce
_8164_pulser_home_move – manual pulser home move

@ Description

_8164_set_pulser_iptmode:

This function is used to configure the input mode of manual pulser.

_8164_pulser_vmove:

As this command is written, the axis begins to move the axis according to manual pulser input. The axis will output one pulse when receive one pulse from pulser, until the ***sd_stop*** or ***emg_stop*** command is written.

_8164_pulser_pmove:

As this command is written, the axis begins to move the axis according to manual pulser input. The axis will output one pulse when receive one pulse from pulser, until the ***sd_stop*** or ***emg_stop*** command is written or the output pulse number reach dist.

_8164_pulser_home_move:

As this command is written, the axis begins to move the axis according to manual pulser input. The axis will output one pulse when receive one pulse from pulser, until the ***sd_stop*** or ***emg_stop*** command is written or the home move finish.

@ Syntax

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

```
I16 _8164_set_pulser_iptmode(I16 AxisNo,I16 InputMode, I16 Inverse);
I16 _8164_pulser_vmove(I16 AxisNo, F64 SpeedLimit);
I16 _8164_pulser_pmove(I16 AxisNo, F64 Dist, F64 SpeedLimit);
I16 _8164_pulser_home_move(I16 AxisNo, I16 HomeType, F64
        SpeedLimit);
```

Visual Basic (Windows 95/NT)

```
B_8164_set_pulser_iptmode (ByVal AxisNo As Integer, ByVal InputMode
        As Integer, ByVal Inverse As Integer) As Integer
B_8164_pulser_vmove (ByVal AxisNo As Integer, ByVal SpeedLimit As
        Double) As Integer
B_8164_pulser_pmove (ByVal AxisNo As Integer, ByVal Dist As Double,
        ByVal SpeedLimit As Double) As Integer
B_8164_pulser_home_move (ByVal AxisNo As Integer, ByVal HomeType
        As Integer, ByVal SpeedLimit As Double) As Integer
```

@ Argument

AxisNo: 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.

Inverse: Reverse the moving direction from pulse direction
Inverse =0, no inverse
Inverse =1, Reverse moving direction

SpeedLimit: The maximum speed in pulser move.
For example, if SpeedLimit is set to be 100 pps, then the axis can move at fastest 100 pps , even the input pulser signal rate is more then 100 pps.

Dist: specified relative distance to move

HomeType: specified home move type
HomeType =0, Command Origin.(that means axis stops when command counter becomes ‘ 0’)
HomeType =1, ORG pin.

@ Return Code

ERR_NoError
ERR_PulserHomeTypeError

6.11 Motion Status

@ Name

_8164_motion_done – Return the motion status

@ Description

_8164_motion_done:

Return the motion status of PCI-8164.

@ Syntax

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

```
I16 _8164_motion_done(I16 AxisNo);
```

Visual Basic (Windows 95/NT)

```
B_8164_motion_done (ByVal AxisNo As Integer) As Integer
```

@ Argument

AxisNo: axis number designated to start manual move

@ Return Value

0	Stop
1	Reserved
2	Reserved
3	Reserved
4	Wait other axis
5	Wait ERC finished
6	Wait DIR Change
7	Backlash compensating
8	Wait PA/PB
9	In home special speed motion
10	In start velocity motion
11	In acceleration
12	In Max velocity motion
13	In deceleration
14	Wait INP
15	Reserved

6.12 Motion Interface I/O

@ Name

_8164_set_alm – Set alarm logic and operating mode
_8164_set_el– Set EL logic and operating mode
_8164_set_inp– Set Inp logic and operating mode
_8164_set_erc– Set ERC logic and timing
_8164_set_servo – Set state of general purpose output pin
_8164_set_sd – Set SD logic and operating mode

@ Description

_8164_set_alm_logic:
Set the active logic of **ALARM** signal input from servo driver. Two reacting modes are available when **ALARM** signal is active.

_8164_set_el:
Set the reacting modes of **EL** signal.

_8164_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.

_8164_set_erc:
You can set the logic and on time of ERC by this function.

_8164_set_servo:
You can set the ON-OFF state of SVON signal by this function. The default value is 1(OFF), which means the SVON is open to GND.

_8164_set_sd_logic:
Set the active logic, latch control and operating mode of **SD** signal input from mechanical system. Users can select whether they want to enable this function. Default state is disabled.

@ Syntax

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

```
I16 _8164_set_alm(I16 AxisNo, I16 alm_logic, I16 alm_mode);
I16 _8164_set_el(I16 AxisNo, I16 el_mode);
I16 _8164_set_inp(I16 AxisNo, I16 inp_enable, I16 inp_logic);
I16 _8164_set_erc(I16 AxisNo, I16 erc_logic, I16 erc_on_time);
I16 _8164_set_servo(I16 AxisNo, I16 on_off);
I16 _8164_set_sd(I16 AxisNo, I16 enable, I16 sd_logic, I16 sd_latch, I16
        sd_mode);
```

Visual Basic (Windows 95/NT)

```
B_8164_set_alm (ByVal AxisNo As Integer, ByVal alm_logic As Integer,
        ByVal alm_mode As Integer) As Integer
B_8164_set_el (ByVal AxisNo As Integer, ByVal el_mode As Integer) As
        Integer
B_8164_set_inp (ByVal AxisNo As Integer, ByVal inp_enable As Integer,
        ByVal inp_logic As Integer) As Integer
B_8164_set_erc (ByVal AxisNo As Integer, ByVal erc_logic As Integer,
        ByVal erc_on_time As Integer) As Integer
B_8164_set_servo (ByVal AxisNo As Integer, ByVal On_Off As Integer) As
        Integer
```

B_8164_set_sd (ByVal AxisNo As Integer, ByVal enable As Integer, ByVal sd_logic As Integer, ByVal sd_latch As Integer, ByVal sd_mode As Integer) As Integer

@ Argument

AxisNo: axis number designated to configure

alm_logic: setting of active logic for ALARM signal

alm_logic=0, active LOW.
alm_logic=1, active HIGH.

alm_mode: reacting modes when receiving ALARM signal.

alm_mode=0, motor immediately stops(Default)
alm_mode=1, motor decelerates then stops.

el_mode: reacting modes when receiving EL signal.

el_mode=0, motor immediately stops.(Default)
el_mode=1, motor decelerates then stops.

inp_enable: INP function enable/disable

inp_enable=0, Disabled (Default)
inp_enable=1, Enabled

inp_logic: setting of active logic for INP signal

inp_logic=0, active LOW.
inp_logic=1, active HIGH.

erc_logic: setting of active logic for ERC signal

erc_logic=0, active LOW.
erc_logic=1, active HIGH.

erc_on_time: Setting of time length of ERC active

erc_on_time=0	12us
erc_on_time=1	102us
erc_on_time=2	409us
erc_on_time=3	1.6ms
erc_on_time=4	13ms
erc_on_time=5	52ms
erc_on_time=6	104ms

on_off: ON-OFF state of SVON signal

on_off = 0 , ON
on_off = 1 , OFF

enable: Enable/disable the SD signal.

enable=0, Disabled (Default)
enable=1, Enabled

sd_logic: setting of active logic for INP signal

sd_logic=0, active LOW.
sd_logic=1, active HIGH.

sd_latch: setting of latch control for SD signal

sd_latch=0, do not latch.
sd_latch=1, latch.

sd_mode: setting the reacting mode of SD signal

sd_mode=0, slow down only
sd_mode=1, slow down then stop

@ Return Code

ERR_NoError

6.13 Motion I/O Monitoring

@ Name

_8164_get_io_status –Get all the motion I/O status of PCI-8164

@ Description

_8164_get_io_status:

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

Bit	Name	Description
0	RDY	RDY pin input
1	ALM	Alarm Signal
2	+EL	Positive Limit Switch
3	-EL	Point Negative Limit Switch
4	ORG	Origin Switch
5	DIR	DIR output
6	Reserved	
7	PCS	PCS signal input
8	ERC	ERC pin output
9	EZ	Index signal
10	Reserved	
11	Latch	Latch signal input
12	SD	Slow Down signal input
13	INP	In-Position signal input
14	SVON	Servo-ON output status

@ Syntax

C/C++ (DOS, Windows 95/98/NT)

I16 _8164_get_io_status(I16 AxisNo, U16 *io_sts);

Visual Basic (Windows 95/NT)

B_8164_get_io_status (ByVal AxisNo As Integer, io_sts As Integer) As Integer

@ Argument

AxisNo: axis number for I/O control and monitoring

***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

6.14 Interrupt Control

@ Name
_8164_int_control – Enable/Disable INT service
_8164_set_int_factor – Set INT factor
_8164_int_enable – Enable event (For Window only)
_8164_int_disable – Disable event (For Window only)
_8164_get_int_status – Get INT Status (For Window only)
_8164_link_interrupt – Set link to interrupt call back function (For Window only)
_8164_get_int_type – Get INT type (For DOS only)
_8164_enter_isr – Enter interrupt service routine (For DOS only)
_8164_leave_isr – Leave interrupt service routine (For DOS only)
_8164_get_event_int – Get event status (For DOS only)
_8164_get_error_int – Get error status (For DOS only)
_8164_get_irq_status – Get IRQ status (For DOS only)
_8164_not_my_irq – Not My IRQ (For DOS only)
_8164_isr0~9, a, b – Interrupt service routine (For DOS only)

@ Description
_8164_int_control:
This function is used to enable interrupt generating to host PC.
_8164_set_int_factor:
This function allows users to select factors to initiate the event int. The error can never be masked once the interrupt service is turn on by _8164_int_control().
The int status of PCI-8164 is composed of two independent parts: **error_int_status** and **event_int_status**. The event_*int_*status recodes the motion and comparator event under **normal operation**, and this kind of INT status can be masked by **_8164_set_int_factor()**. The error_int_status is for abnormal stop of PCI-8164, for example: EL, ALM …etc. This kind of INT cannot be masked. The following is the definition of these two int_status. By setting the relative bit as 1, PCI-8164 can generate INT signal to host PC.

Bit	Description
0	Normal Stop
1	Next command continued
2	Continuous pre-register is empty and allow users to fill new command
3	(Reserved)
4	Acceleration Start
5	Acceleration End
6	Deceleration Start
7	Deceleration End
8	(Reserved)
9	(Reserved)

10	Step-losing occur
11	General Comparator compared
12	Compared triggered for axis 0,1
13	(Reserved)
14	Latched for axis2,3
15	ORG on
16	SD on
17~31	(Reserved)

_8164_int_enable : (For **Window** only.)

This function is used to assign the window INT event.

_8164_int_disable: (For **Window** only.)

This function is used to disable the window INT event.

_8164_get_int_status: (For **Window** only.)

This function allows user to identify what cause the interrupt signal. After user gets this value, the status register will be cleared to 0. The return value is two 32 bits unsigned integers. The first one is for error_int_status, which is not able to mask by _8164_set_int_factor(). The definition for bit of error_int_status is as following:

error_int_status : can not be masked	
Bit	Interrupt Factor
0	+SL Stop
1	-SL Stop
2	(Reserved)
3	General Comparator Stop
4	(Reserved)
5	+EL
6	-EL
7	ALM
8	(Reserved)
9	(Reserved)
10	SD on then stop
11~31	Reserved

The second is for event_int_status, which can be masked by _8164_set_int_factor(). The definition for bit of event_int_status is as following:

event_int_status : can be masked by function call ***_8164_int_*****factor()**	
Bit	Description
0	Normal Stop
1	Next command continued
2	Continuous pre-register is empty and allow users to fill new command
3	(Reserved)

4	Acceleration Start
5	Acceleration End
6	Deceleration Start
7	Deceleration End
8	(Reserved)
9	(Reserved)
10	Step-losing occur
11	General Comparator compared
12	Compared triggered for axis 0,1
13	(Reserved)
14	Latched for axis2,3
15	ORG on
16	SD on
17~31	(Reserved)

_8164_link_interrupt: (For **Window** only.)
This function is used to link interrupt call back function.

_8164_get_int_type: **(** This function is for **DOS** only **)**
This function is used to detect which kind of INP occurred.

_8164_enter_isr: **(** This function is for **DOS** only **)**
This function is used to inform system that process is now entering interrupt service routine.

_8164_leave_isr: **(** This function is for **DOS** only **)**
This function is used to inform system that process is now leaving interrupt service routine.

_8164_get_event_int: **(** This function is for **DOS** only **)**
This function is used to get event_int_status.

_8164_get_error_int: **(** This function is for **DOS** only **)**
This function is used to get error_int_status.

_8164_get_irq_status: **(** This function is for **DOS** only **)**
This function allows user to confirm if the designated card generates the INT signal to host PC.

_8164_not_my_irq: **(** This function is for **DOS** only **)**
This function must be called after knowing not the designated card generates the INT signal to host PC.

_8164_isr0, _8164_isr1, _8164_isr2, _8164_isr3, ... _8164_isr9, _8164_isra, _8164_isrb: **(** Theses function is for **DOS** only **)**
Individual Interrupt service routine for card 0~11.

@ Syntax

C/C++ (DOS)

```
I16 _8164_int_control(U16 cardNo, U16 intFlag );
I16 _8164_set_int_factor(I16 AxisNo, U32 int_factor );
I16  _8164_get_int_type(I16 AxisNo, U16 *int_type);
I16  _8164_enter_isr(I16 AxisNo);
I16  _8164_leave_isr(I16 AxisNo);
I16  _8164_get_event_int(I16 AxisNo, U32 *event_int);
```

```
I16  _8164_get_error_int(I16 AxisNo, U32 *error_int);
I16  _8164_get_irq_status(U16 cardNo, U16 *sts);
I16  _8164_not_my_irq(I16 CardNo);
void interrupt _8164_isr0 (void);
void interrupt _8164_isr1 (void);
void interrupt _8164_isr2 (void);
void interrupt _8164_isr3 (void);
void interrupt _8164_isr4 (void);
void interrupt _8164_isr5 (void);
void interrupt _8164_isr6 (void);
void interrupt _8164_isr7 (void);
void interrupt _8164_isr8 (void);
void interrupt _8164_isr9 (void);
void interrupt _8164_isra (void);
void interrupt _8164_isrb (void);
```

C/C++ (Windows 95/98/NT)

```
I16 _8164_int_control(U16 cardNo, U16 intFlag );
I16 _8164_set_int_factor(I16 AxisNo, U32 int_factor );
I16 _8164_int_enable(I16 CardNo, HANDLE *phEvent);
I16 _8164_int_disable(I16 CardNo);
I16 _8164_get_int_status(I16 AxisNo, U32 *error_int_status, U32
        *event_int_status );
I16 _8164_link_interrupt(I16 CardNo, void ( __stdcall *callbackAddr)(I16
        IntAxisNoInCard));
```

Visual Basic (Windows 95/NT)

```
B_8164_int_control (ByVal CardNo As Integer, ByVal intFlag As Integer) As
        Integer
B_8164_set_int_factor (ByVal AxisNo As Integer, ByVal int_factor As Long)
        As Integer
B_8164_int_enable (ByVal CardNo As Integer, phEvent As Long) As
        Integer
B_8164_int_disable (ByVal CardNo As Integer) As Integer
B_8164_get_int_status (ByVal AxisNo As Integer, error_int_status As Long,
        event_int_status As Long) As Integer
B_8164_link_interrupt (ByVal CardNo As Integer, ByVal lpCallBackProc As
        Long) As Integer
```

@ Argument

cardNo: card number 0,1,2,3…
AxisNo: axis number 0,1,2,3,4…
intFlag: int flag, 0 or 1 (0: Disable, 1:Enable)
int_factor: interrupt factor, refer to previous table
***int_type:** Interrupt type, (1: error int, 2: event int, 3: both happened)
***event_int:** event_int_status, , refer to previous table
***error_int:** error_int_status, refer to previous table
***sts:** (0: not this card' s IRQ, 1: this card' s IRQ)
***phEvent:** event handler (Windows)
***error_int_status**: refer to previous table
***event_int_status:** refer to previous table

@ Return Code

ERR_NoError

ERR_EventNotEnableYet
ERR_LinkIntError
ERR_CardNoErrot

6.15 Position Control and Counters

@ Name

_8164_get_position – Get the value of feedback position counter
_8164_set_position – Set the feedback position counter
_8164_get_command – Get the value of command position counter
_8164_set_command – Set the command position counter
_8164_get_error_counter – Get the value of position error counter
_8164_reset_error_counter – Reset the position error counter
_8164_get_general_counter – Get the value of general counter
_8164_set_general_counter – Set the general counter
_8164_get_target_pos – Get the value of target position recorder
_8164_reset_target_pos – Reset target position recorder
_8164_get_rest_command – Get remaining pulse till end of motion
_8164_check_rdp – Get the ramping down point data

@ Description

_8164_get_position():

This function is used to read the value of feedback position counter. Note, this value has already been processed by move ratio. If move ratio is 0.5, than the value read will be twice as the counter value. The source of feedback counter is selectable by function **_8164_set_feedback_src()** to be external EA/EB or pulse output of PCI-8164.

_8164_set_position():

This function is used to change the feedback position counter to the specified value. Note, the value to be set will be processed by move ratio. If move ratio is 0.5, than the set value will be twice as given value.

_8164_get_command():

This function is used to read the value of command position counter. The source of command position counter is the pulse output of PCI-8164.

_8164_set_command():

This function is used to change the value of command position counter.

_8164_get_error_counter():

This function is used to read the value of position error counter.

_8164_reset_error_counter():

This function is used to clear position error counter.

_8164_get_general_counter():

This function is used to read the value of general counter.

_8164_set_general_counter():

This function is used to set the counting source of and change the value of general counter. (By default, the source is pulser input.)

_8164_get_target_pos():

This function is used to read the value of target position recorder. The target position recorder is maintained by PCI-8164 software driver. It records the position to settle down for current running motion.

_8164_reset_target_pos():

This function is used to set new value for target position recorder. It is necessary to call this function when home return completion or when new feedback counter value is set by function _8164_set_position().

_8164_get_rest_command():

This function is used to read remaining pulse counts till end of current motion.

_8164_check_rdp():

This function is used to read the ramping down point data. The ramping down point is the position where deceleration starts. The data is stored as pulse count, and it cause the axis start to decelerate when remaining pulse count reach the data.

@ Syntax

C/C++ (DOS, Windows 95/98/NT)

```
I16 _8164_get_position(I16 AxisNo, F64 *pos);
I16 _8164_set_position(I16 AxisNo, F64 pos);
I16 _8164_get_command(I16 AxisNo, I32 *cmd);
I16 _8164_set_command(I16 AxisNo, I32 cmd);
I16 _8164_get_error_counter(I16 AxisNo, I16 *error_counter);
I16 _8164_reset_error_counter(I16 AxisNo);
I16 _8164_get_general_counter(I16 AxisNo, F64 *CntValue);
I16 _8164_set_general_counter(I16 AxisNo,I16 CntSrc, F64 CntValue);
I16 _8164_get_target_pos(I16 AxisNo, F64 *T_pos);
I16 _8164_reset_target_pos(I16 AxisNo, F64 T_pos);
I16 _8164_get_rest_command(I16 AxisNo, I32 *rest_command);
I16 _8164_check_rdp(I16 AxisNo, I32 *rdp_command);
```

Visual Basic (Windows 95/NT)

```
B_8164_get_position (ByVal AxisNo As Integer, Pos As Double) As Integer
B_8164_set_position (ByVal AxisNo As Integer, ByVal Pos As Double) As
        Integer
B_8164_get_command (ByVal AxisNo As Integer, cmd As Long) As Integer
B_8164_set_command (ByVal AxisNo As Integer, ByVal cmd As Long) As
        Integer
B_8164_get_error_counter (ByVal AxisNo As Integer, error_counter As
        Integer) As Integer
B_8164_reset_error_counter (ByVal AxisNo As Integer) As Integer
B_8164_get_general_counter (ByVal AxisNo As Integer, CntValue As
        Double) As Integer
B_8164_set_general_counter (ByVal AxisNo As Integer, ByVal CntSrc As
        Integer, ByVal CntValue As Double) As Integer
B_8164_get_target_pos (ByVal AxisNo As Integer, Pos As Double) As
        Integer
B_8164_reset_target_pos (ByVal AxisNo As Integer, ByVal Pos As Double)
        As Integer
B_8164_get_rest_command (ByVal AxisNo As Integer, rest_command As
        Long) As Integer
```

B_8164_check_rdp (ByVal AxisNo As Integer, rdp_command As Long) As Integer

@ Argument

AxisNo: Axis number
Pos, *Pos: Feedback position counter value,
range: -134217728~134217727
cmd, *cmd: Command position counter value,
range: -134217728~134217727
error_counter, *error_counter: Position error counter value,
range: -32768~32767
T_pos, *T_pos: Target position recorder value,
T_ range: -134217728~134217727
CntValue, * CntValue: General counter value,
range: -134217728~134217727
rest_command, *rest_command: Rest pulse count till end,
range: -134217728~134217727
rdp_command, *rdp_command: Ramping down point data
range: 0~167777215
CntSrc: Source of general counter
0 : command
1: EA/EB
2: PA/PB (Default)
3: CLK/2

@ Return Code

ERR_NoError
ERR_PosOutofRange

6.16 Position Compare and Latch

@ Name

_8164_set_ltc_logic – Set the LTC logic
_8164_get_latch_data – Get latched counter data
_8164_set_soft_limit – Set soft limit
_8164_enable_soft_limit – Enable soft limit function
_8164_disable_soft_limit – Disable soft limit function
_8164_set_error_counter_check – Step-losing detection setup
_8164_set_general_comparator – Set general-purposed comparator
_8164_set_trigger_comparator – Set trigger comparator
_8164_set_trigger_type – Set the trigger output type
_8164_check_compare_data – Check current comparator data
_8164_check_compare_status – Check current comparator status
_8164_set_auto_compare – Set comparing data source for auto loading
_8164_build_compare_function – Build compare data via constant interval
_8164_build_compare_table – Build compare data via compare table
_8164_cmp_v_change – Speed change by comparator

@ Description

_8164_set_ltc_logic():

This function is used to set the logic of latch input. This function is applicable only for last two axes in every PCI-8164 card.

_8164_get_latch_data():

After the latch signal arrived, this function is used to read the latched value of counters.

_8164_set_soft_limit():

This function is used to set the value of soft limit.

_8164_enable_soft_limit(),_8164_disable_soft_limit():

These two functions are used to enable/disable the soft limit function. Once enabled, the action of soft limit will be exactly the same as physical limit.

_8164_set_error_counter_check():

This function is used to enable the step losing checking facility. By giving an tolerance value, the PCI-8164 will generate an interrupt (event_int_status , bit 10) when position error counter exceed tolerance.

_8164_set_general_comparator():

This function is used to set the source and comparing value for general comparator. When the source counter value reached the comparing value, the PCI-8164 will generate an interrupt (event_int_status , bit 11).

_8164_set_trigger_comparator():

This function is used to set the comparing method and value for trigger comparator. When the feedback position counter value reached the comparing value, the PCI-8164 will generate trigger a pulse output via **CMP** and an interrupt (event_int_status , bit 12) will also be sent to host PC. If _8164_set_auto_compare is used, then comparing value set by this function will be ignored automatically. *Note: it is applicable only for first two axes in every PCI-8164 card.*

_8164_set_trigger_type():

This function is used to set the trigger output mode, one shot pulse or lifting & keeping high.

_8164_check_compare_data():

This function is used to get current comparing data of designated comparator.

_8164_check_compare_status():

This function is used to get status of all comparator. When some comparator comes into existence, the relative bit of cmp_sts will become ‘ 1’ , otherwise ‘ 0’ .

_8164_set_auto_compare():

This function is used to set the comparing data source of trigger comparator. The source can be either a function or a table.

_8164_build_compare_function():

This function is used to build comparing function by defining the start / end point and interval. There is no limitation on the max number of comparing data.

Note: Please turn off all interrupt function, when FIFO is used.

_8164_build_compare_table():

This function is used to build comparing table by defining data array. The size of array is limited to 1024.

Note: Please turn off all interrupt function, when FIFO is used.

_8164_cmp_v_change():

This function is used to setup comparator velocity change function. It is in fact a V_change function but acts when general comparator comes into existence. When this function is issued, the parameter “CmpAction” of **_8164_set_general_comparator()** must be set ‘ 3’ . The compare data is also set by **_8164_set_general_comparator()**. While, the remain distance, the compare point’ s velocity , the new velocity and the acceleration time are set by **_8164_cmp_v_change()**.

@ Syntax

C/C++ (DOS, Windows 95/98/NT)

```
I16 _8164_set_ltc_logic(I16 AxisNo_2or3, I16 ltc_logic);
I16 _8164_get_latch_data(I16 AxisNo, I16 Counter, F64 *Pos);
I16 _8164_set_soft_limit(I16 AxisNo, I32 PLimit, I32 NLimit);
I16 _8164_disable_soft_limit(I16 AxisNo);
I16 _8164_enable_soft_limit(I16 AxisNo, I16 Action);
I16 _8164_set_error_counter_check(I16 AxisNo, I16 Tolerance, I16
        On_Off);
I16 _8164_set_general_comparator(I16 AxisNo, I16 CmpSrc, I16
```

CmpMethod, I16 CmpAction, F64 Data);
I16 _8164_set_trigger_comparator(I16 AxisNo, I16 CmpMethod, F64 Data);
I16 _8164_set_trigger_type(I16 AxisNo, I16 TriggerType);
I16 _8164_check_compare_data(I16 AxisNo, I16 CmpSrc, F64 *Pos);
I16 _8164_check_compare_status(I16 AxisNo, U16 *cmp_sts);
I16 _8164_set_auto_compare(I16 AxisNo ,I16 SelectSrc);
I16 _8164_cmp_v_change(I16 AxisNo, F64 Res_dist, F64 oldvel, F64 newvel, F64 AccTime)

C/C++ (Windows 95/98/NT)

I16 _8164_build_compare_function(I16 AxisNo, F64 Start, F64 End, F64 Interval, I16 Device);
I16 _8164_build_compare_table(I16 AxisNo, F64 *TableArray, I16 Size, I16 Device);

C/C++ (Dos)

I16 _8164_build_compare_function(I16 AxisNo, F64 Start, F64 End, F64 Interval);
I16 _8164_build_compare_table(I16 AxisNo, F64 *TableArray, I16 Size);

Visual Basic (Windows 95/NT)

B_8164_set_ltc_logic (ByVal AxisNo As Integer, ByVal ltc_logic As Integer) As Integer
B_8164_get_latch_data (ByVal AxisNo As Integer, ByVal Counter As Integer, Pos As Double) As Integer
B_8164_set_soft_limit (ByVal AxisNo As Integer, ByVal PLimit As Long, ByVal NLimit As Long) As Integer
B_8164_disable_soft_limit (ByVal AxisNo As Integer) As Integer
B_8164_enable_soft_limit (ByVal AxisNo As Integer, ByVal Action As Integer) As Integer
B_8164_set_error_counter_check (ByVal AxisNo As Integer, ByVal Tolerance As Integer, ByVal On_Off As Integer) As Integer
B_8164_set_general_comparator (ByVal AxisNo As Integer, ByVal CmpSrc As Integer, ByVal CmpMethod As Integer, ByVal CmpAction As Integer, ByVal Data As Double) As Integer
B_8164_set_trigger_comparator (ByVal AxisNo As Integer, ByVal CmpMethod As Integer, ByVal Data As Double) As Integer
B_8164_set_trigger_type (ByVal AxisNo As Integer, ByVal TriggerType As Integer) As Integer
B_8164_check_compare_data (ByVal AxisNo As Integer, ByVal CmpSrc As Integer, Pos As Double) As Integer
B_8164_check_compare_status (ByVal AxisNo As Integer, cmp_sts As Integer) As Integer
B_8164_set_auto_compare (ByVal AxisNo As Integer, ByVal SelectSrc As Integer) As Integer
B_8164_build_compare_function (ByVal AxisNo As Integer, ByVal Start As Double, ByVal End As Double, ByVal Interval As Double, ByVal Device As Integer) As Integer
B_8164_build_compare_table (ByVal AxisNo As Integer, TableArray As Double, ByVal Size As Integer, ByVal Device As Integer) As Integer
B_8164_cmp_v_change(ByVal AxisNo, ByVal Res_dist as Double, ByVal oldvel as Double, ByVal newvel as Double, ByVal AccTime as

Double)

@ Argument

AxisNo_2or3: Axis number, for last two axes in one card
ltc_logic: 0 means active low, 1 means active high
AxisNo: Axis number
Counter: Specified Counter
Counter = 0 , Command counter
Counter = 1 , Feedback counter
Counter = 2 , Error Counter
Counter = 3 , General Counter
Pos: Latched counter value,
PLimit: Soft limit value in positive direction
NLimit: Soft limit value in negative direction
Action: The reacting method of soft limit
Action =0, INT only
Action =1, Immediately stop
Action =2, slow down then stop
Action =3, reserved
Tolerance: The tolerance of step-losing detection
On_Off: Enable / Disable step-losing detection
On_Off =0, Disable
On_Off =1, Enable
CmpSrc: The comparing source counter
CmpSrc =0, Command Counter
CmpSrc =1, Feedback Counter
CmpSrc =2, Error Counter
CmpSrc =3, General Counter
CmpMethod: The comparing method
CmpMethod =0, No compare
CmpMethod =1, CmpValue=Counter (Directionless)
CmpMethod =2, CmpValue=Counter (+Dir)
CmpMethod =3, CmpValue=Counter (-Dir)
CmpMethod =4, CmpValue>Counter
CmpMethod =5, CmpValue<Counter
CmpAction: The reacting mode when comparison comes into exist
CmpAction =0, INT only
CmpAction =1, Immediately stop
CmpAction =2, slow down then stop
CmpAction =3, speed change
Data: Comparing value,
TriggerType: Selection of type of trigger output mode
TriggerType =0, one shoot
TriggerType =1, high
cmp_sts: status of comparator

Bit	Meaning
0	+Softlimit On
1	-SoftLimit On
2	Error counter comparator On
3	General comparator On
4	Trigger comparator On (for 0 , 1 axis only)

SelectSrc: The comparing data source

SelectSrc =0, disable auto compare
SelectSrc =1, use FIFO
SelectSrc =2, use compare function **(Window only)**
SelectSrc =3, use compare table value **(Window only)**

Start: Start point of compare function
End: End point of compare function
Interval: Interval of compare function
TableArray: Array of comparing data
Size: Size of table array
Device: Selection of reload device for comparator data
Device =0, RAM & interrupt (windows only)
Device =1, FIFO
Res_dist: The remain distance from the compare point. After comparison, the original target position will be ignored, and the axis will keep moving the Res_dist.
oldvel: The velocity at compare point. User must specify it manually.
newvel: The new velocity.
AccTime: The acceleration time.

@ Return Code

ERR_NoError
ERR_CompareNoError
ERR_CompareMethodError
ERR_CompareAxisError
ERR_CompareTableSizeError
ERR_CompareFunctionError
ERR_CompareTableNotReady
ERR_CompareLineNotReady
ERR_HardwareCompareAxisWrong
ERR_AutocompareSourceWrong
ERR_CompareDeviceTypeError

6.17 Continuous motion

@ Name
_8164_set_continuous_move – Enable continuous motion
_8164_check_continuous_buffer – check if the buffer is empty

@ Description
_8164_set_continuous_move():
This function is necessary before and after continuous motion.
_8164_check_continuous_buffer():
This function is used to detect if the command buffer is empty or not. Once the command buffer is empty, user may write next motion command into it. Otherwise (said, not empty), the new command will overwrite previous in buffer.

@ Syntax

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

```
I16 _8164_set_continuous_move(I16 AxisNo, I16 conti_flag);
I16 _8164_check_continuous_buffer(I16 AxisNo);
```

Visual Basic (Windows 95/NT)

```
B_8164_set_continuous_move (ByVal AxisNo As Integer, ByVal conti_flag
        As Integer) As Integer
B_8164_check_continuous_buffer (ByVal AxisNo As Integer) As Integer
```

@ Argument
AxisNo: axis number designated
conti_flag: Flag for continuous motion
conti_flag = 0, one-shoot motion, end of continuous motion
conti_flag = 1, continuous motion, start of continuous motion

@ Return Value
ERR_NoError

Return **value of _8164_check_continuous_buffer()**:
0: Continuous buffer is empty
1: Continuous buffer is full

6.18 Multiple Axes Simultaneous Operation

@ Name

_8164_set_tr_move_all – Multi-axis simultaneous operation setup.
_8164_start_move_all – Begin a multi-axis trapezoidal profile motion
_8164_stop_move_all –Simultaneously stop Multi-axis motion

@ Description

Theses functions are related simultaneous operation of multi-axis even in different cards. The simultaneous multi-axis operation means to start or stop moving specified axes at the same time. The move axes are specified by parameter **"AxisArray"** and the number of axes are defined by parameter "***TotalAxes"*** in **_8164_set_tr_move_all()**.

When properly setup with **_8164_set_tr_move_all()**, the function **_8164_start_move_all()** will cause all specified axes to begin trapezoidal relative moving, and _8164_stop_move_all() will stop them. Both functions guarantee that motion Start/Stop on all specified axes at the same time. **Note** that it is necessary to make connections according to Section 3.14 on CN3 if these two functions are needed.

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 (simultaneous motion).

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

_8164_set_tr_move_all(2, axes, dist, str_vel, max_vel, Tacc, Tdec);
_8164_start_move_all(axes[0]);

return   ERR_NoError;
}
```

@ Syntax

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

```
I16 _8164_set_tr_move_all(I16 TotalAxes, I16 *AxisArray, F64 *DistA, F64
        *StrVelA, F64 *MaxVelA, F64 *TaccA, F64 *TdecA);
```

```
I16 _8164_start_move_all(I16 FirstAxisNo);
I16 _8164_stop_move_all(I16 FirstAxisNo);
```

Visual Basic (Windows 95/NT)

```
B_8164_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_8164_start_move_all(ByVal FirstAxisNo  As Integer);
B_8164_stop_move_all(ByVal FirstAxisNo As Integer);
```

@ Argument

TotalAxes: number of axes for simultaneous motion, 1~48.
*** AxisArray**: specified axes number array designated to move.
*** DistA**: 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:** deceleration time array in unit of second
FirstAxisNo : the first element in AxisArray.

@ Return Code

ERR_NoError
ERR_SpeedError

6.19 General-purposed TTL output

@ Name
_8164_d_output – Digital Output
_8164_get_dio_status – Get DIO status

@ Description
_8164_d_output():
Set the on_off status for general-purposed TTL Digital output pin.
_8164_get_dio_status():
Read status of all digital output pin.

@ Syntax

C/C++ (DOS, Windows 95/NT)
```
I16 _8164_d_output(I16 CardNo, I16 Ch_No, I16 value);
I16 _8164_get_dio_status(I16 CardNo, U16 *dio_sts);
```
Visual Basic (Windows 95/NT)
```
B_8164_d_output (ByVal CardNo As Integer, ByVal Ch_No As Integer,
        ByVal value As Integer) As Integer
B_8164_get_dio_status (ByVal CardNo As Integer, dio_sts As Integer) As
        Integer
```

@ Argument
CardNo: Designated card number
Ch_No: Designated channel number 0~5
Value: On-Off Value for output
Value =0, output OFF
Value =1, output ON
dio_status: Digital output status
bit0~bit5 for channel 0~5 , respectively

@ Return Value
ERR_NoError
ERR_DioNoError

7

Connection Example

This chapter shows some connection examples between PCI-8164 and servo drivers and stepping drivers.

7.1 General Description of Wiring

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

CN2 :Main connection between PCI-8164 and pulse input servo driver or stepping driver.

CN3: Receive pulse command from manual pulser.

CN4: Connector for simultaneously start or stop multiple PCI-8164 cards.

CN5: TTL digital output.

Figure 7.1 shows how to integrate PCI-8164 with a physical system.

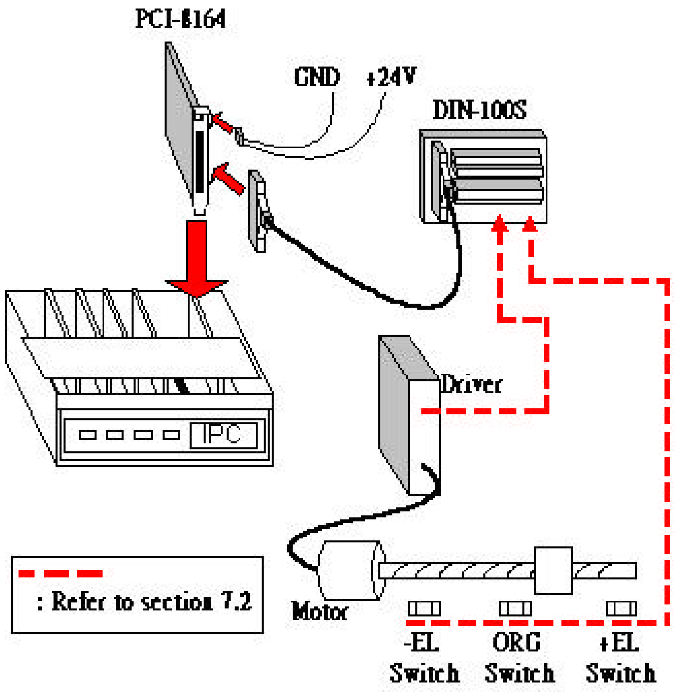


Figure 7.1 System Integration with PCI-8164

7.2 Connection Example with Servo Driver

In this section, we use ***Panasonic Servo Driver*** as an example to show how to connect it with ***PCI-8164***. Figure 7.2 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 1X **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-8164, user should set the operating mode to **Position** Mode.

Wiring of PCI-8164 with Panasonic MSD

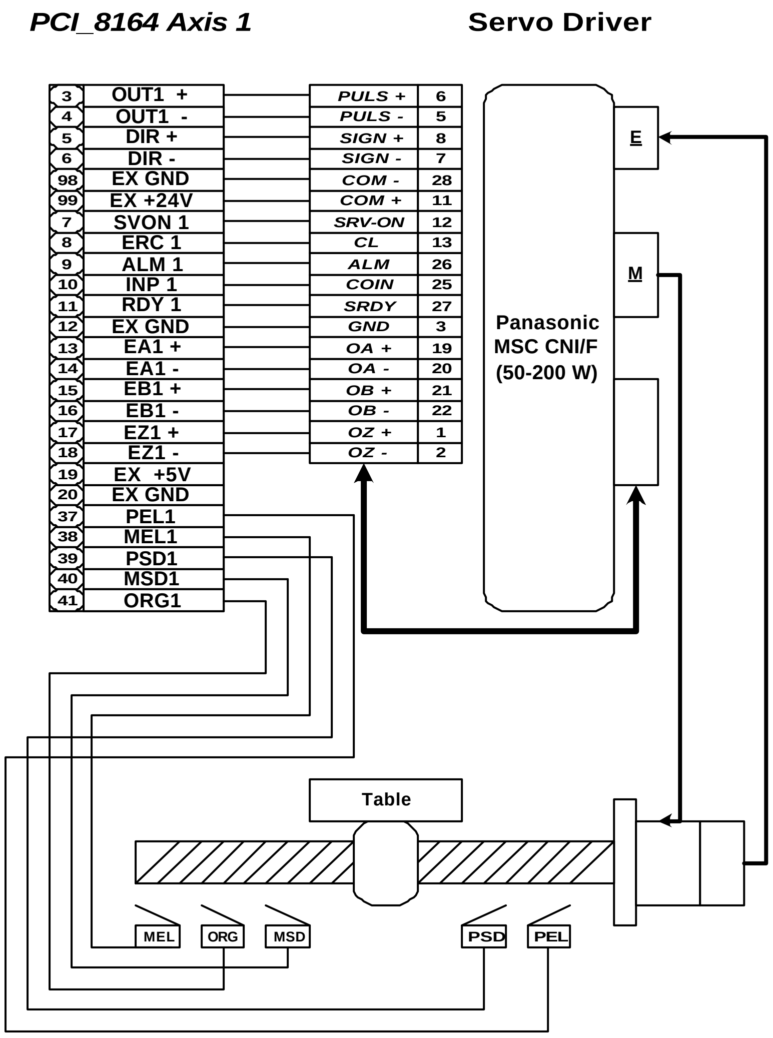


Figure 7.2 Connection of PCI-8164 with Panasonic Driver

Wiring of PCI-8164 with SANYO AC Servo PY2

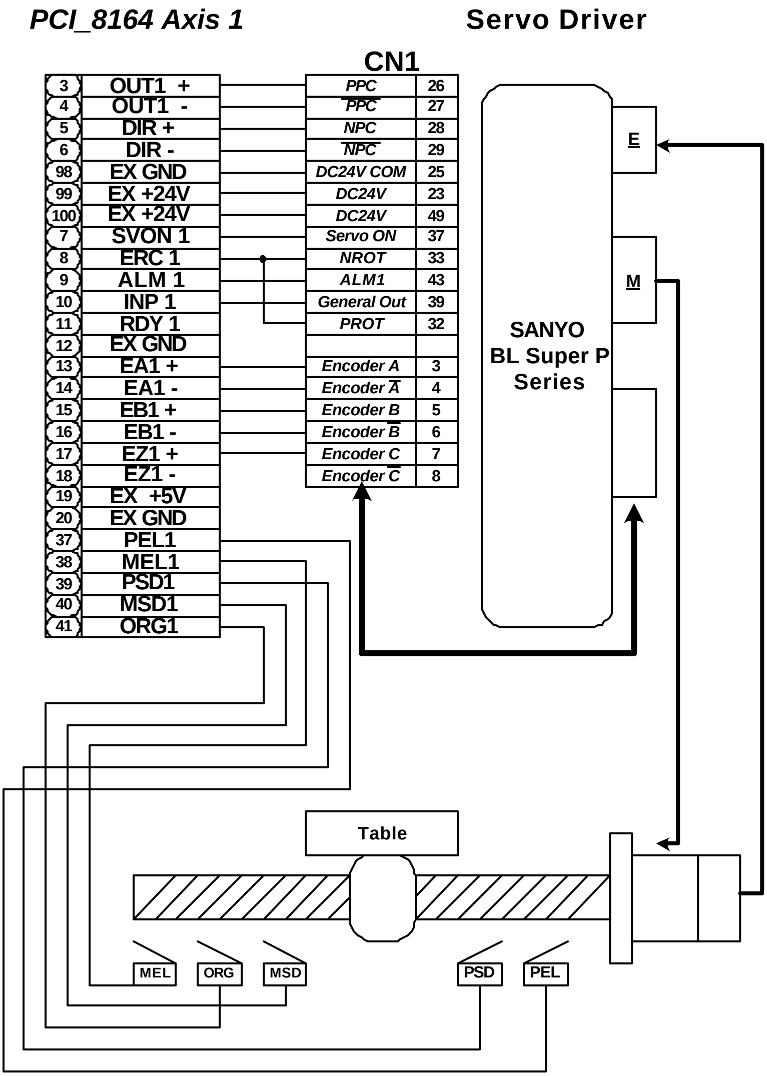


Figure 7.3 Connection of PCI-8164 with SANYO Driver

Product Warranty/Service

Seller warrants that equipment furnished will be free form defects in material and workmanship for a period of one year from the confirmed date of purchase of the original buyer and that upon written notice of any such defect, Seller will, at its option, repair or replace the defective item under the terms of this warranty, subject to the provisions and specific exclusions listed herein.

This warranty shall not apply to equipment that has been previously repaired or altered outside our plant in any way as to, in the judgment of the manufacturer, affect its reliability. Nor will it apply if the equipment has been used in a manner exceeding its specifications or if the serial number has been removed.

Seller does not assume any liability for consequential damages as a result from our products uses, and in any event our liability shall not exceed the original selling price of the equipment.

The equipment warranty shall constitute the sole and exclusive remedy of any Buyer of Seller equipment and the sole and exclusive liability of the Seller, its successors or assigns, in connection with equipment purchased and in lieu of all other warranties expressed implied or statutory, including, but not limited to, any implied warranty of merchant ability or fitness and all other obligations or liabilities of seller, its successors or assigns.

The equipment must be returned postage-prepaid. Package it securely and insure it. You will be charged for parts and labor if you lack proof of date of purchase, or if the warranty period is expired.