PCI-8164/MPC-8164/PXI-8164
Advanced 4-Axis Servo/Stepper Motion Control Card
User's Manual

Manual Rev. 2.00

Revision Date: August 25, 2006

Part No: 50-11124-1050

Recycled Paper

Advance Technologies; Automate the World.

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

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

Trademarks

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

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

Getting service from ADLINK

Customer satisfaction is top priority for ADLINK TECHNOLOGY INC. Please contact us should you require any service or assistance.

ADLINK TECHNOLOGY INC.

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

Email or fax this completed service form for prompt and satisfactory service.

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

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

Table of Contents

List of Tables

List of Figures

1 Introduction

The PCI-/MPC-/PXI-8164 is an advanced 4-axis motion controller card that generates high frequency pulses (6.55 MHz) to drive stepper or servomotors, and provides a 2-axis circular interpolation, 4-axis linear interpolation, or continuous interpolation for continual velocity. The PCI-/MPC-/PXI-8164 also changes position and/or speed on the fly with a single axis operation.

Multiple PCI-/MPC-/PXI-8164 cards may be installed in one system. Incremental encoder interface on all four axes provide the ability to correct positioning errors generated by inaccurate mechanical transmissions. With the aid of an onboard FIFO, the PCI-/MPC-/PXI-8164 also performs precise and extremely fast position comparison and trigger functions without compromising CPU resources. In addition, a mechanical sensor interface, a servo motor interface, and general-purposed I/O signals are provided for easy system integration.

The following figures show the functional block diagrams of the 8164 card in PCI, MPC, and PXI interfaces. The PCI-/MPC-/PXI-8164 uses one ASIC (PCL6045) to perform all 4 axes motion controls. The motion control functions include linear and S-curve acceleration/deceleration, circular interpolation between two axes, linear interpolation between 2-4 axes, continuous motion positioning, and more than 13 home return modes. All these functions and complex computations are performed internally by the ASIC, thus minimizing CPU usage and eliminating real-time issues.

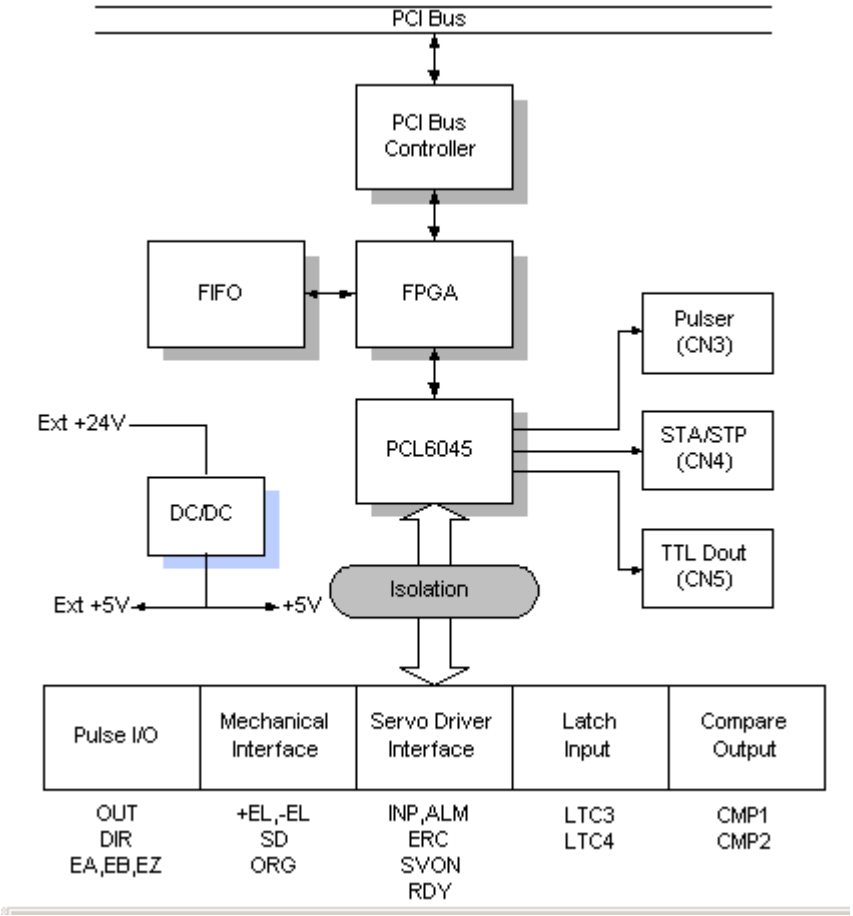


Figure 1-1: PCI-8164 block diagram

The MPC-8164 is an advanced 4-axis motion controller card with a PC104 interface. All features and specification are leveraged with the PCI-8164, except for some differences in the user I/O interfaces. Figure 1-2 shows the MPC-8164 card block diagram.

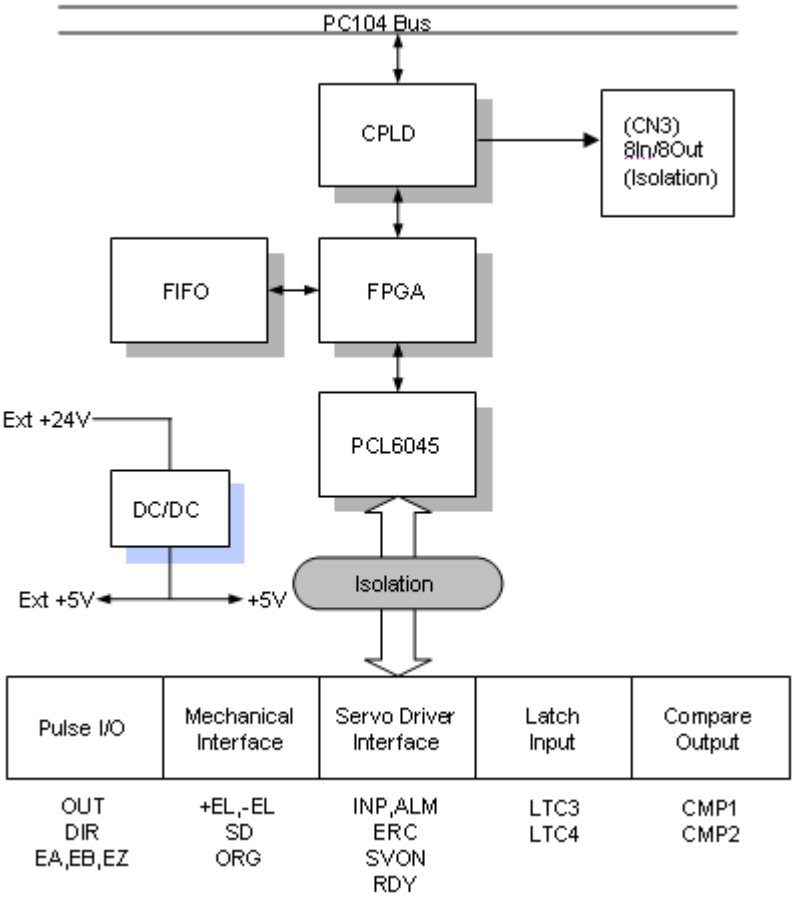


Figure 1-2: MPC-8164 block diagram

The PXI-8164 is an advanced 4-axis motion controller card with a PXI interface. All features and specification are the same with the PCI-8164, except for some differences in the user I/O interfaces. Figure 1-3 shows the PXI-8164 the block diagram.

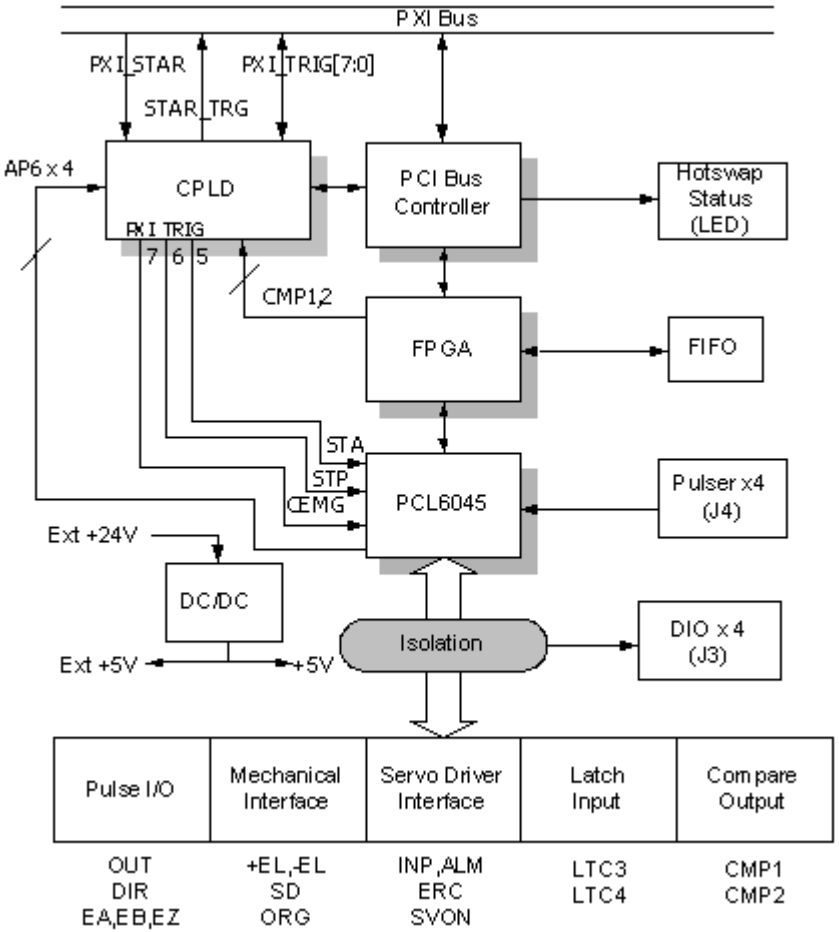


Figure 1-3: PXI-8164 block diagram

Motion Creator is a Windows-based application development software package that comes with the card. Motion Creator is useful for debugging a motion control system during the design phase of a project. An on-screen display lists all installed axes information and the card's I/O signal status.

DOS and Windows programming libraries for C++ and Visual Basic are included together with some sample programs to illustrate the operations of these functions.

Figure 1-4 illustrates a flow chart of application development using the contents of this manual. Refer to the related chapters for details.

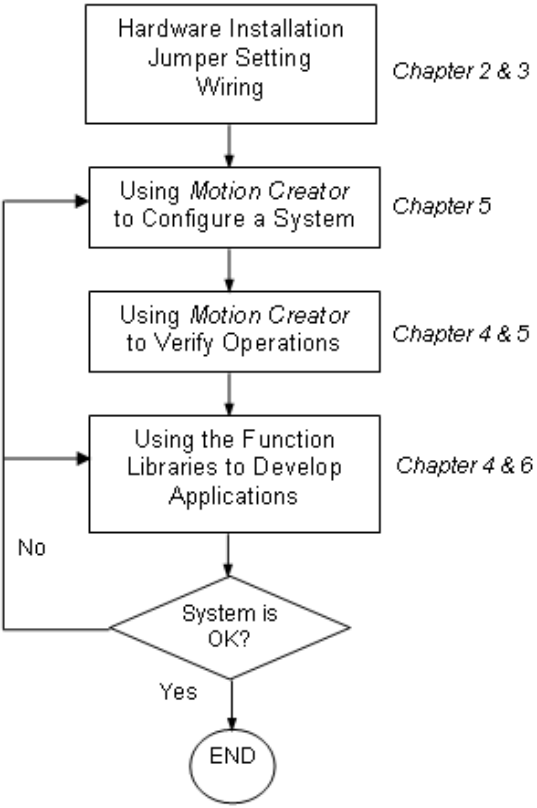


Figure 1-4: Application building flow chart

1.1 Features

1.1.1 PCI-8164

- 32-bit PCI bus, Plug and Play
- 4 axes of step and direction pulse output for controlling stepping or servomotor
- 6.55 MPPS maximum output frequency
- OUT/DIR, CW/CCW pulse output options
- Programmable acceleration and deceleration time for all modes
- Trapezoidal and S-curve velocity profiles for all modes
- Any 2 of 4 axes circular interpolation
- Any 2-4 of 4 axes linear interpolation
- Continuous interpolation for contour following motion
- Change position and speed on the fly
- Change speed by condition comparing
- 13 home return modes with auto searching
- Hardware backlash compensator and vibration suppression
- 2 software end-limits for each axis
- 28-bit up/down counter for incremental encoder feedback
- Home switch, index signal (EZ), positive, and negative end limit switches interface on all axes
- 2-axis high speed position latch input
- 2-axis position compare trigger output with 4k FIFO auto-loading
- 2500V_{rms} isolated digital input and output signals
- Programmable interrupt sources
- Simultaneous start/stop motion on multiple axes
- Manual pulser input interface (A small steering device that generates pulses when turned)
- Software supports a maximum of up to 12 PCI-8164 cards (48 axes) operation in one system
- Compact, half-sized PCB

- Includes *Motion Creator*, a Microsoft Windows-based application development software
- Libraries and utilities support DOS, Windows® 9X/NT/2000/XP, and Linux

1.1.2 MPC-8164

- 16-bit PC104 bus
- 4 axes of step and direction pulse output for controlling stepping or servomotor
- 6.55 MPPS maximum output frequency
- OUT/DIR, CW/CCW pulse output options
- Programmable acceleration and deceleration time for all modes
- Trapezoidal and S-curve velocity profiles for all modes
- Any 2 of 4 axes circular interpolation
- Any 2-4 of 4 axes linear interpolation
- Continuous interpolation for contour following motion
- Change position and speed on the fly
- Change speed by comparator condition
- 13 home return modes with auto searching
- Hardware backlash compensator and vibration suppression
- 2 Software end-limits for each axis
- 28-bit up/down counter for incremental encoder feedback
- Home switch, index signal (EZ), positive, and negative end limit switches interface on all axes
- 2-axis high speed position latch input
- 2-axis position compare trigger output with 4k FIFO auto-loading
- 2500_{Vrms} isolated digital input and output signals
- Programmable interrupt sources
- 8 channels of general purpose photo-isolated digital inputs
- 8 channels of general purpose open collector digital outputs
- Software supports a maximum of up to 4 MPC-8164 cards (16 axes) operation in one system
- Includes *Motion Creator*, a Microsoft Windows-based application development software
- Libraries and utilities support DOS, Windows® 98/NT/2000/XP, and Windows® XP/NT Embedded
- Libraries for Linux and Windows® CE systems

1.1.3 PXI-8164

- PXI specifications Rev. 2.0-compliant
- Multiple modules synchronized via PXI trigger bus
- 3U Eurocard form factor, CompactPCI compliant (PICMG 2.0 R2.1)
- 4-CH isolated digital I/O
- 4 axes of step and direction pulse output for controlling stepping or servomotor
- 6.55 MPPS maximum output frequency
- OUT/DIR, CW/CCW pulse output options
- Programmable acceleration and deceleration time for all modes
- Trapezoidal and S-curve velocity profiles for all modes
- Any 2 of 4 axes circular interpolation
- Any 2-4 of 4 axes linear interpolation
- Continuous interpolation for contour following motion
- Change position and speed on the fly
- Change speed by condition comparing
- 13 home return modes with auto searching
- Hardware backlash compensator and vibration suppression
- 2 software end-limits for each axis
- 28-bit up/down counter for incremental encoder feedback
- Home switch, index signal (EZ), positive, and negative end limit switches interface on all axes
- 2-axis high speed position latch input
- 2-axis position compare trigger output with 4k FIFO auto-loading
- Programmable interrupt sources
- Simultaneous start/stop motion on multiple axes
- Manual pulser input interface (A small steering device that generates pulses when turned)
- Software supports a maximum of up to 12 PXI-8164 cards (48 axes) operation in one system

- Includes *Motion Creator*, a Microsoft Windows-based application development software
- Libraries and utilities DOS, Windows® 9x/NT/2000/XP, and Linux

1.2 Specifications

Applicable motors

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

Performance

- 4 controllable axes
- 6.55MPPS maximum pulse output frequency, linear, trapezoidal, or S-Curve velocity profile drive
- 19.66 MHz internal reference clock
- 28-bit up/down counter range: 0 to 268,435,455 or –134,217,728 to +134,217,727
- Position pulse setting range (28-bit): -134,217,728 to +134,217,728
- Pulse rate setting ranges (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 signals

- Input/Output signals for each axis
- Opto-isolated digital input with 2500V_{rms} isolation voltage
- OUT and DIR command pulse output pins
- EA and EB incremental encoder signals input pins
- EZ encoder index signal input pin
- ±EL, SD/PCS, and ORG mechanical limit/switch signal input pins
- INP, ALM, and ERC servomotor interface I/O pins
- LTC position latch input pin
- CMP position compare output pin
- SVON general-purposed digital output pin
- RDY general-purposed digital input pin
- PA and PB (PCI-8164/PXI-8164) pulse signal input pin
- STA and STP (PCI-8164/PXI-8164) simultaneous start/stop signal

General-purpose output

- 6 TTL level digital outputs (PCI-8164 only)
- 8 digital inputs/8 digital outputs (MPC-8164 only)
- 4 digital inputs/4 digital outputs (PXI-8164 only)

General specifications

- 100-pin SCSI-type connector
- Operating temperature: 0ºC - 50ºC
- Storage temperature: -20ºC - 80ºC
- 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

Dimensions

- PCI-8164: 185 mm (L) X 106.68 mm (W)
- MPC-8164: 152 mm (L) X 104.7 mm (W)
- PXI-8164: 3U Eurocard form factor, CompactPCI-compliant (PICMG 2.0 R2.1)

1.3 Software support

1.3.1 Programming library

Programming libraries for MS-DOS and Borland C/C++ (Version 3.1) and DLLs for Windows® 95/98/NT/2000/XP come bundled with the PCI-8164/PXI-8164 card package. Support for Linux-based systems is also included.

MPC-8164 supports DOS/Windows® 98/NT/2000/XP, Windows® XP/NT Embedded, Windows® CE, and Linux.

1.3.2 Motion Creator

This Windows-based utility sets up cards, motors, and systems. It also debugs hardware and software problems and enables the user to set I/O logic parameters that can be loaded in their own programs. Refer to Chapter 5 for more details.

2 Installation

Follow these steps to install the PCI-/MPC-/PXI-8164 card.

- Check the card package contents (section 2.1)
- Check the card PCB and face plate/front panel layout (section 2.2)
- Install the card to the chassis (section 2.3)
- Install the drivers (section 2.4)
- Refer to the I/O signal connections (chapter 3) and their operation (chapter 4)
- Refer to the connector pin assignments (the remaining sections) and wiring connections

2.1 Package contents

Check the package contents for the following items:

- PCI-8164/MPC-8164/PXI-8164 card
- ADLINK All-in-one CD
- +24V power input cable (for CN1) accessories (PCI-8164 only)
- Optional terminal board for wiring purposes

If any of these items are missing or damaged, contact your dealer immediately. Save the original packaging for future shipment.

2.2 PCI-8164 layout

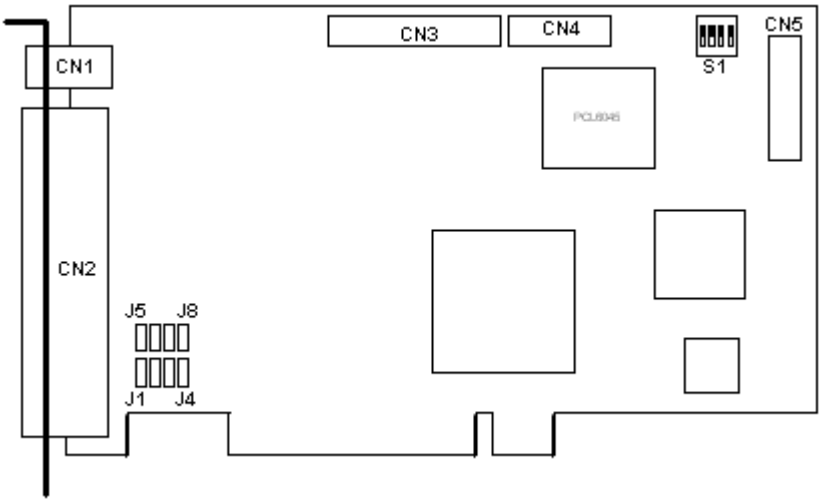


Figure 2-1: PCI-8164 PCB layout

CN1:	External Power Input Connector
CN2:	Input / Output Signal Connector
CN3:	Manual Pulse Signal Connector
CN4:	Simultaneous Start / Stop Connector
CN5:	General purpose TTL output
S1:	End limit logic selection switch
J1-J8:	Pulse output selection jumper

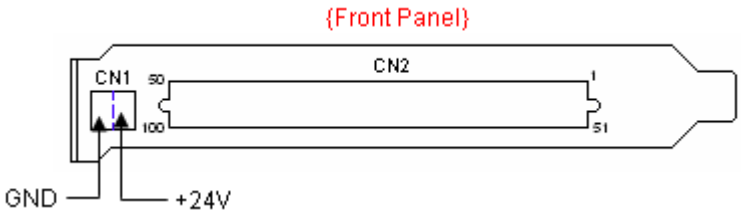


Figure 2-2: PCI-8164 face plate

2.3 MPC-8164 layout

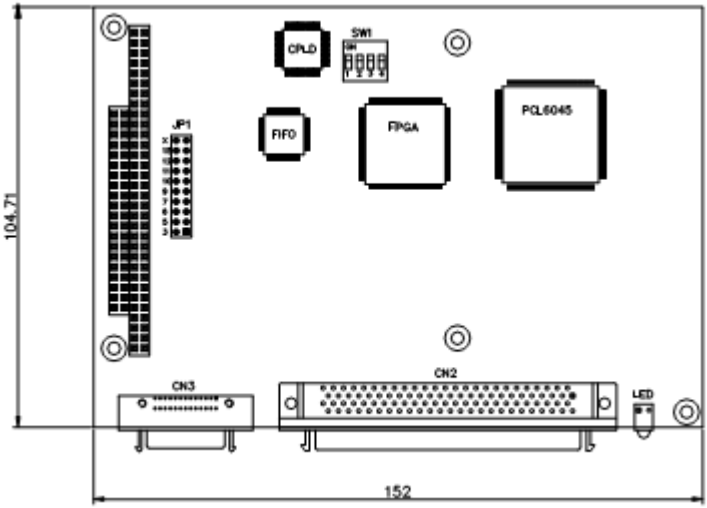


Figure 2-3: MPC-8164 PCB layout

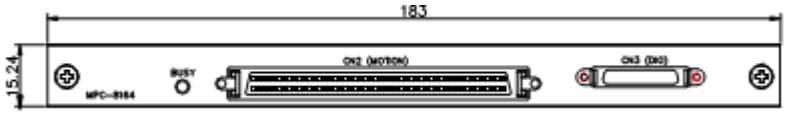


Figure 2-4: MPC-8164 face plate

CN2: Input / Output Signal Connector
CN3: 8 DI / 8 DO Connector
JP1: IRQ selection
SW1: Base Address Selection

2.4 PXI-8164 layout

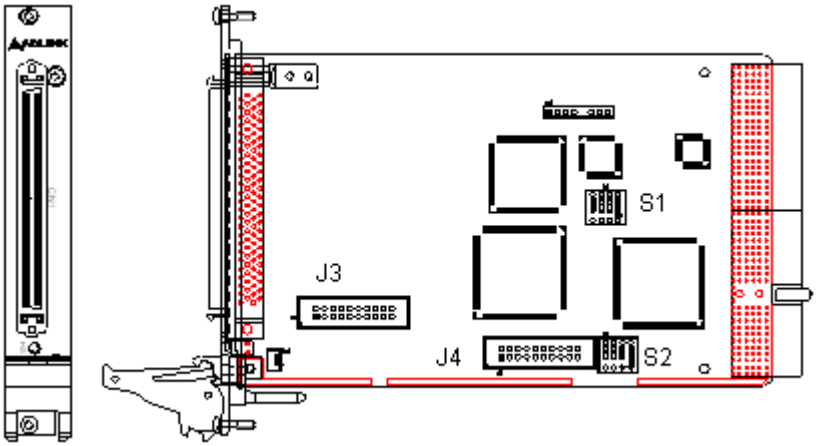


Figure 2-5: PXI-8164 layout and front panel

S1: Switch setting for EL logic

S2: Card ID setting from 0-11

J3: 4-CH isolated digital Input/output

J4: 4-axis pulser input interface

2.5 PCI-8164/PXI-8164 hardware installation

2.5.1 Hardware configuration

Since the PCI-8164/PXI-8164 card is Plug and Play, the memory allocation (I/O port locations) and the IRQ channel are automatically assigned by the system BIOS. The address assignment is done on a board-by-board basis for all PCI cards installed in the system.

2.5.2 PCI slot selection

The PCI-8164 card may be installed in any available PCI slot. The PXI-8164 card may be installed in any PXI slot.

CAUTION Do not install the PCI card into a PC/AT (ISA) slot.

2.5.3 Installing the PCI-8164 card

1. Discharge any static buildup from your body by touching the metal case of the computer. Hold the card on its edges and avoid touching the components.
2. Set the card jumper(s) according to your requirements.
3. Turn off the computer and all connected peripherals, then open the computer chassis.
4. Locate a 32-bit PCI slot. PCI slots are shorter than ISA or EISA slots and usually comes in white or ivory.
5. Remove the metal bracket opposite the slot you want to use. Keep the bracket screw for later use.
6. Insert the PCI card connectors (golden fingers) to the slot, then press firmly until the card is properly seated on the slot.
7. Secure the card with the bracket screw you removed earlier, then replace the computer chassis.
8. Connect all peripherals, then turn the computer on.

Installation notes

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

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

2.5.4 Installing the PXI-8164 card

1. Follow steps 1 to 2 of the previous section.
2. Select an available PXI slot, then remove the metal cover opposite the slot you want to use. Keep the metal cover and screws for later use.
3. Align the card's top and bottom edges with the chassis card guides, then carefully slide it into the chassis.
4. Lift the card ejector latch until it locks in place.
5. Secure the card with two screws.
6. Connect all peripherals, then turn the computer on.

2.6 MPC-8164 hardware installation

2.6.1 Hardware configuration

The MPC-8164 card is PC104-compliant. The onboard DIP switches and jumpers assign the card's I/O port locations and IRQ channels.

A single-board setup has a default setting of 0x200 and IRQ5. In GEME systems, the default value varies depending on the location of the card. Refer to the following table:

GEME level	Base address	IRQ
1	0x300	9
2	0x200	5
3	0x280	10

Table 2-1: GEME hardware configuration

Base address setting

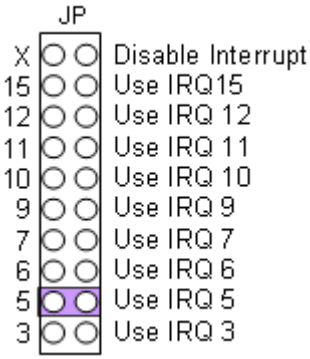
The base address is set by SW1 (pins 2 to 4). Note that pin 1 is reserved. If all DIPs are set to OFF, the base address is 0x200. Default settings are dependent on the order.

DIP Switch (2 3 4)	Base Address	DIP Switch (2 3 4)	Base Address
1 1 1	0x3C0	1 1 0	0x2C0
0 1 1	0x380	0 1 0	0x280
1 0 1	0x340	1 0 0	0x240
0 0 1	0x300	0 0 0	0x200

Table 2-2: Base Addresses

IRQ setting

The JP1 setting assigns the IRQ channel.



Installation note

Make sure that the system has an aqvailable I/O address and IRQ channel for the card. If there are none available, adjust the card I/O address and IRQ channel to empty.

2.7 Driver installation

PCI-8164/PXI-8164

1. Place the ADLINK All-In-One CD to the CD-ROM drive.
2. When the Autorun screen appears, select **Driver Installation** > **Motion Control** > **PCI-8164/PXI-8164**.
3. Follow screen procedures to install, then restart the system after installation is completed.

NOTE When using MS-DOS, install the drivers from the \Motion Control\PCI-8164\DOS_BC directory of the CD.

MPC-8164

1. Place the ADLINK All-In-One CD to the CR-ROM drive.
2. When the Autorun screen appears, select **Driver Installation** > **Motion Control** > **MPC-8164**.
3. Launch the MPC-8164 Add/Remove utililty from the Start menu or installed directory to register the new card. The I/O address and IRQ channel must be the same with the settings on the board.
4. Restart the computer.

NOTES When using MS-DOS, install the drivers from the \Motion Control\MPC-8164\DOS_BC directory of the CD.

You may also download the latest software from the ADLINK website (www.adlinktech.com).

2.8 CN1 pin assignments: External Power Input (PCI-8164 only)

CN1 Pin No	Name	Description
1	EGND	External power ground
2	CN1_24V	+24V DC ± 5% External power supply

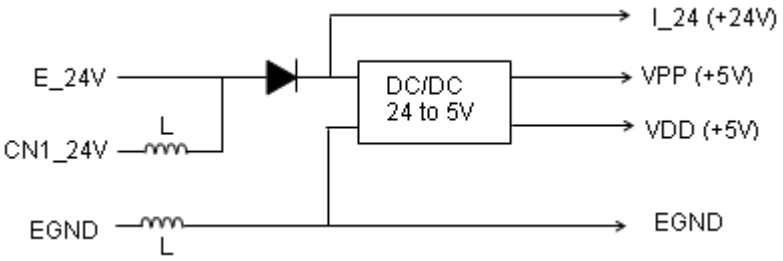
NOTES

- CN1 is a plug-in terminal board with no screws.
- Use the external power supply. A +24V DC is used by external input/output signal circuits. The power circuit configuration is shown below.
- Wires for connection to CN1:
  - Solid wire: φ0.32 mm to φ0.65 mm (AWG28 to AWG22)
  - Twisted wire: 0.08 mm^2 to 0.32 mm^2 (AWG28 to AWG22)
  - Naked wire length: 10 mm standard

The diagram below shows the external power supply system of the PCI-8164. An external +24V power must be provided. An onboard regulator generates +5V for both internal and external use.

CAUTION The output current capacity of the +5V power source from the onboard DC/DC is limited. DO NOT use this to drive several devices simultaneously, especially stepper motors or external encoders.

NOTE MPC-8164 and PXI-8164 do not have the CN1 for power input. Use the E_24V and EGND pins of CN2. L is an inductor for EMI use.



2.9 CN3 pin assignments: Manual Pulse Input (PCI-8164 only)

CN3 is for the manual pulse input.

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

NOTE The PCI bus provides the signals for the VCC and DGND pins. These signals are not isolated.

2.10 J4 pin assignments: Manual Pulse Input
(PXI-8164 only)

No.	Name	Function	No.	Name	Function
1	DGND	Bus power ground	2	PB4	Axis 3 Pulser PHB
3	PA4	Axis 4 Pulser PHA	4	PB3	Axis 2 Pulser PHB
5	PA3	Axis 3 Pulser PHA	6	VCC	Bus Power +5V
7	DGND	Bus power ground	8	PB2	Axis 1 Pulser PHB
9	PA2	Axis 1 Pulser PHA	10	PB1	Axis 0 Pulser PHB
11	PA1	Axis 0 Pulser PHA	12	VCC	Bus Power +5V
13	--	N/A	14	--	N/A
15	--	N/A	16	--	N/A
17	--	N/A	18	--	N/A
19	--	N/A	20	--	N/A

2.11 CN3 pin assignments: General Purpose DIO
(MPC-8164 only)

Pin No	Signal Name	Pin No	Signal Name
1	DOCOM	2	DOCOM
3	DOCOM	4	DOCOM
5	DO0	6	DO1
7	DO2	8	DO3
9	DO4	10	DO5
11	DO6	12	DO7
13	--	14	DICOM
15	DICOM	16	DICOM
17	DICOM	18	DI0
19	DI1	20	DI2
21	DI3	22	DI4
23	DI5	24	DI6
25	DI7	26	--

2.12 J3 pin assignments: Isolated DIO
(PXI-8164 only)

No.	Name	Function	No.	Name	Function
1	DICOM	Digital In Common	2	DOCOM	Digital Out Common
3	DI0	Input Channel 0	4	DO0	Output Channel 0
5	DI1	Input Channel 1	6	DO1	Output Channel 1
7	DICOM	Digital In Common	8	DOCOM	Digital Out Common
9	DI2	Input Channel 2	10	DO2	Output Channel 2
11	DI3	Input Channel 3	12	DO3	Ouput Channel 3
13	DICOM	Digital In Common	14	DOCOM	Digital Out Common
15	--	N/A	16	--	N/A
17	--	N/A	18	--	N/A
19	--	N/A	20	--	N/A

2.13 CN2 pin assignments: Main Connector

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

No.	Name	I/O	Function (axis /)	No.	Name	I/O	Function (axis /)
1	VPP	O	+5V power supply output	51	VPP	O	+5V power supply output
2	EGND		Ext. power ground	52	EGND		Ext. power ground
3	OUT1+	O	Pulse signal (+),	53	OUT3+	O	Pulse signal (+),
4	OUT1-	O	Pulse signal (-),	54	OUT3-	O	Pulse signal (-),
5	DIR1+	O	Dir. signal (+),	55	DIR3+	O	Dir. signal (+),
6	DIR1-	O	Dir. signal (-),	56	DIR3-	O	Dir. signal (-),
7	SVON1	O	Multi-purpose signal,	57	SVON3	O	Multi-purpose signal,
8	ERC1	O	Dev. ctr, clr. signal,	58	ERC3	O	Dev. ctr, clr. signal,
9	ALM1	I	Alarm signal,	59	ALM3	I	Alarm signal,
10	INP1	I	In-position signal,	60	INP3	I	In-position signal,
11	RDY1	I	Multi-purpose signal,	61	RDY3	I	Multi-purpose signal,
12	EGND		Ext. power ground	62	EGND		Ext. power ground
13	EA1+	I	Encoder A-phase (+),	63	EA3+	I	Encoder A-phase (+),
14	EA1-	I	Encoder A-phase (-),	64	EA3-	I	Encoder A-phase (-),
15	EB1+	I	Encoder B-phase (+),	65	EB3+	I	Encoder B-phase (+),
16	EB1-	I	Encoder B-phase (-),	66	EB3-	I	Encoder B-phase (-),
17	EZ1+	I	Encoder Z-phase (+),	67	EZ3+	I	Encoder Z-phase (+),
18	EZ1-	I	Encoder Z-phase (-),	68	EZ3-	I	Encoder Z-phase (-),
19	VPP	O	+5V power supply output	69	VPP	O	+5V power supply output
20	EGND		Ext. power ground	70	EGND		Ext. power ground
21	OUT2+	O	Pulse signal (+),	71	OUT4+	O	Pulse signal (+),
22	OUT2-	O	Pulse signal (-),	72	OUT4-	O	Pulse signal (-),
23	DIR2+	O	Dir. signal (+),	73	DIR4+	O	Dir. signal (+),
24	DIR2-	O	Dir. signal (-),	74	DIR4-	O	Dir. signal (-),
25	SVON2	O	Multi-purpose signal,	75	SVON4	O	Multi-purpose signal,
26	ERC2	O	Dev. ctr, clr. signal,	76	ERC4	O	Dev. ctr, clr. signal,
27	ALM2	I	Alarm signal,	77	ALM4	I	Alarm signal,
28	INP2	I	In-position signal,	78	INP4	I	In-position signal,
29	RDY2	I	Multi-purpose signal,	79	RDY4	I	Multi-purpose signal,
30	EGND		Ext. power ground	80	EGND		Ext. power ground
31	EA2+	I	Encoder A-phase (+),	81	EA4+	I	Encoder A-phase (+),
32	EA2-	I	Encoder A-phase (-),	82	EA4-	I	Encoder A-phase (-),
33	EB2+	I	Encoder B-phase (+),	83	EB4+	I	Encoder B-phase (+),
34	EB2-	I	Encoder B-phase (-),	84	EB4-	I	Encoder B-phase (-),
35	EZ2+	I	Encoder Z-phase (+),	85	EZ4+	I	Encoder Z-phase (+),
36	EZ2-	I	Encoder Z-phase (-),	86	EZ4-	I	Encoder Z-phase (-),
37	PEL1	I	End limit signal (+),	87	PEL3	I	End limit signal (+),
38	MEL1	I	End limit signal (-),	88	MEL3	I	End limit signal (-),
39	CMP1	O	Position compare output	89	LTC3	I	Position latch input

40	SD/PCS1	I	Ramp-down signal	90	SD/PCS3	I	Ramp-down signal
41	ORG1	I	Origin signal,	91	ORG3	I	Origin signal,
42	EGND		Ext. power ground	92	EGND		Ext. power ground
43	PEL2	I	End limit signal (+),	93	PEL4	I	End limit signal (+),
44	MEL2	I	End limit signal (-),	94	MEL4	I	End limit signal (-),
45	CMP2	O	Position compare output	95	LTC4	I	Position latch input,
46	SD/PCS2	I	Ramp-down signal	96	SD/PCS4	I	Ramp-down signal
47	ORG2	I	Origin signal,	97	ORG4	I	Origin signal,
48	EGND		Ext. power ground	98	GND		Ext. power ground
49	EGND		Ext. power ground	99	E_24V		Ext. power supply, +24V
50	EGND		Ext. power ground	100	E_24V		Ext. power supply, +24V

2.14 CN4 pin assignments: Simultaneous Start/Stop (PCI-8164 only)

CN4 is for simultaneous start/stop signals for multiple axes or multiple cards.

No.	Name	Function (Axis)
1	DGND	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	VCC	Bus power output, +5V

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

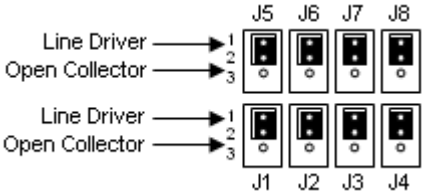
2.15 CN5 pin assignment: TTL Output (PCI-8164 only)

CN5 is 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.	Not used

2.16 Jumper setting for pulse output (PCI-8164 only)

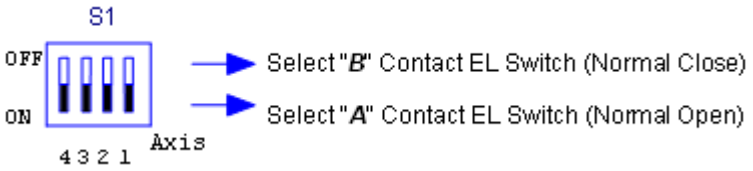
J1 - J8 sets the type of pulse output signals (DIR and OUT). The output signal type may either be differential line driver or open collector output. Refer to section 3.1 for detailed jumper settings. The default setting is differential line driver mode.



2.17 Switch setting for EL Logic

The S1 switch sets the EL limit switching type. By default the EL switch is set to ON, which is the “normally open” position (or "A" contact type), while OFF is the “normally closed” position (or “B” contact type).

For safety reasons, you must set a type that will make the end-limit active when it is broken or disconnected.



NOTE MPC-8164 uses a software function for this setting.

2.18 CN3 pin assignment: General Purpose DI/DO ports (MPC-8164 only)

CN3 Pin No	Signal Name	CN3 Pin No	Signal Name
1	DOCOM	2	DOCOM
3	DOCOM	4	DOCOM
5	DO0	6	DO1
7	DO2	8	DO3
9	DO4	10	DO5
11	DO6	12	DO7
13	--	14	DICOM
15	DICOM	16	DICOM
17	DICOM	18	DI0
19	DI1	20	DI2
21	DI3	22	DI4
23	DI5	24	DI6
25	DI7	26	--

2.19 S2 card ID switch setting (PXI-8164 only)

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

NOTE Other settings are invalid. In order to enable this function, see section 6.21.

3 Signal Connections

This chapter describes the signal connections of the card I/Os. Refer to the contents of this chapter before wiring any cables between the card and any motor drivers.

This chapter contains the following sections:

3.1 Pulse Output Signals OUT and DIR

The PCI-/MPC-/PXI-8164 has 4 axis pulse output signals. Each axis has two pairs of OUT and DIR signals to transmit the pulse train and to indicate the direction. The OUT and DIR signals may also be programmed as CW and CCW signal pairs. Refer to section 4.1.1 for details of the logical characteristics of the OUT and DIR signals. This section details the electrical characteristics of the OUT and DIR signals. Each signal consists of a pair of differential signals. For example, OUT2 consists of OUT2+ and OUT2- signals. The following table shows all pulse output signals on CN2.

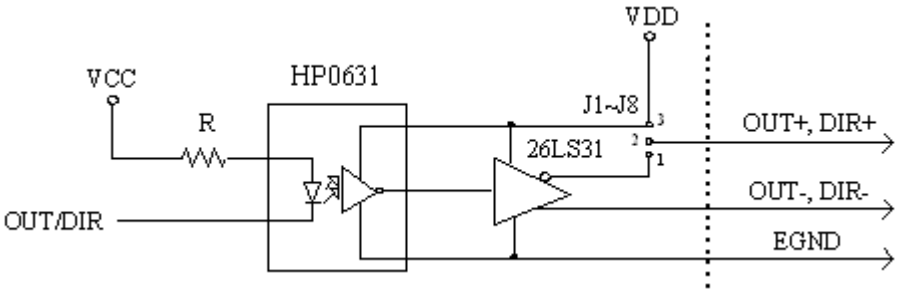
CN2 Pin No.	Signal Name	Description	Axis #
3	OUT1+	Pulse signals (+)	1
4	OUT1-	Pulse signals (-)	1
5	DIR1+	Direction signal (+)	1
6	DIR1-	Direction signal (-)	1
21	OUT2+	Pulse signals (+)	2
22	OUT2-	Pulse signals (-)	2
23	DIR2+	Direction signal (+)	2
24	DIR2-	Direction signal (-)	2
53	OUT3+	Pulse signals (+)	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 differential line drivers or open collector output. For PCI-8164 card, you can select the output mode by closing either breaks between 1 and 2 or 2 and 3 of jumpers J1-J8.

Output Signal	For differential line driver output, close breaks between 1 and 2 of:	For open collector output, close breaks 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

By default, the OUT and DIR are set to differential line driver mode.

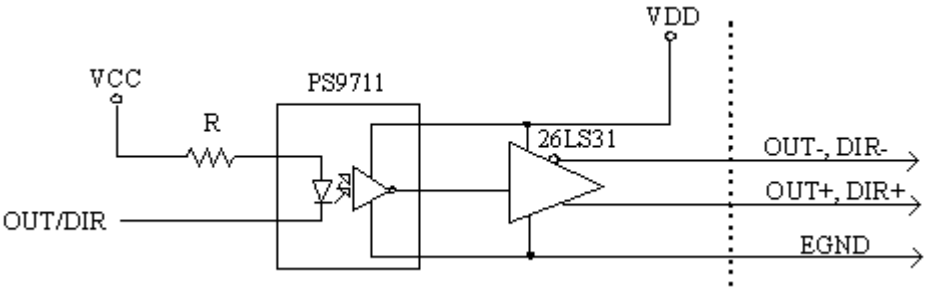
The wiring diagram below illustrates the OUT and DIR signals on the 4 axes of PCI-8164 card.



NOTE When the pulse output is set to open collector output mode, OUT- and DIR- transmits OUT signals. The sink current must not exceed 20 mA on the OUT- and DIR- pins. By default, pin 1-2 of the jumper is shorted.

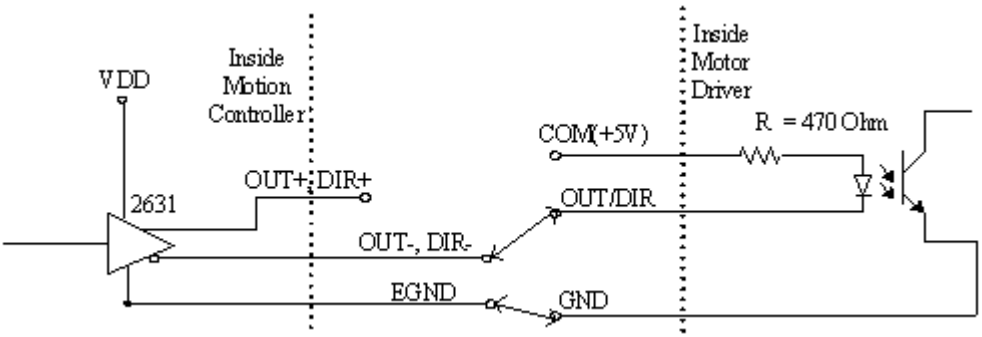
USAGE Short pin 2-3 of the jumper and connect OUT+/DIR+ to a 470 ohm pulse input interface's COM of driver. See the following figure.

MPC-8164/PXI-8164



Non-differential type wiring example (MPC-8164/PXI-8164, or PCI-8164 when pin 2-3 of the jumper is shorted)

Choose one of OUT/DIR+ and OUT/DIR- to connect to the driver's OUT/DIR.



WARNING The sink current must not exceed 20 mA to prevent damage to the PCI-/MPC-/PXI-8164 card!

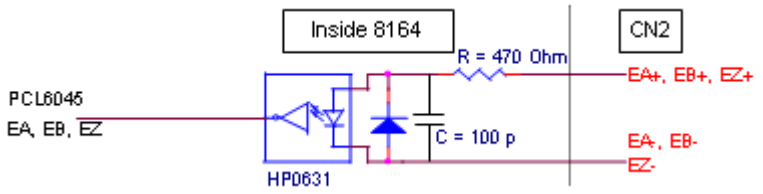
3.2 Encoder Feedback Signals EA, EB and EZ

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

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 diagram below shows the input circuit of the EA, EB, and EZ signals.



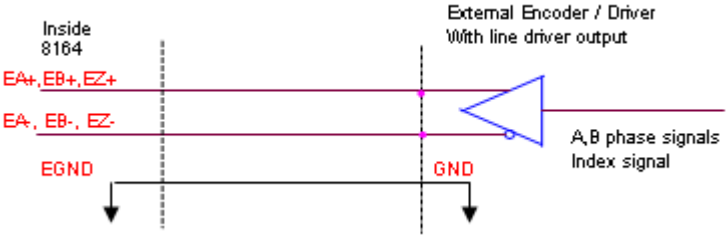
Note that the voltage across each differential pair of encoder input signals (EA+, EA-), (EB+, EB-), and (EZ+, EZ-) should be at least 3.5V. Therefore, the output current must be observed when connecting to the encoder feedback or motor driver feedback to avoid

over driving the source. The differential signal pairs are converted to digital signals EA, EB, and EZ, then feed to the PCL6045 ASIC.

Below are examples of input signal connection with an external circuit. The input circuit may be connected to an encoder or motor driver if it is equipped with a differential line driver or an open collector output.

Connection to line driver output

To drive the card encoder input, the driver output must provide at least 3.5V across the differential pairs with at least 6 mA driving capacity. The grounds of both sides must be tied together. The maximum frequency will be 4 Mhz or more depending on the wiring distance and signal conditioning.

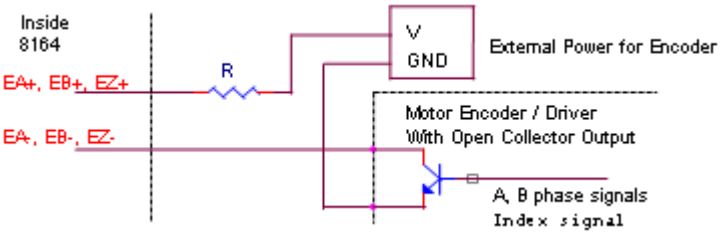


Connection to open collector output

You need an external power supply to connect with an open collector output. Some motor drivers provide the power source. The diagram below shows the connection between the card, encoder, and the power supply. Note that an external current limiting resistor R is necessary to protect the card's input circuit. The following table lists the suggested resistor values according to the encoder power supply.

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

I_f = 6 mA max



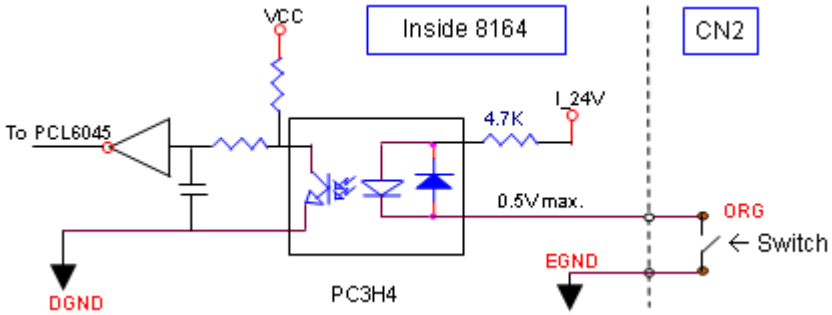
For more operation information on the encoder feedback signals, refer to section 4.4.

3.3 Origin Signal ORG

The origin signals (ORG1-ORG4) are used as input signals for the origin of the mechanism. The table below lists signal names, pin numbers, and axis numbers.

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

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



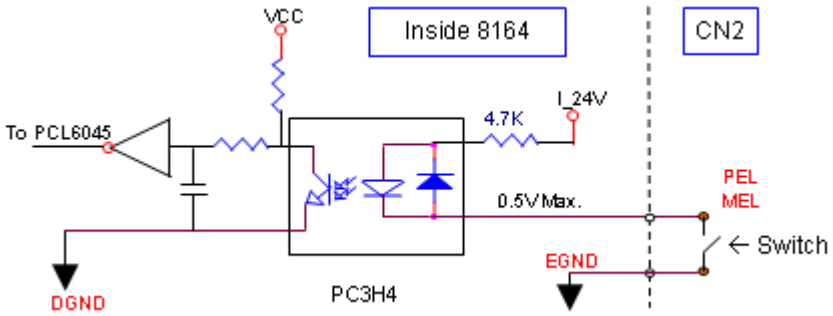
When the motion controller is operated in the home return mode, the ORG signal is used to inhibit the control output signals (OUT and DIR). For detailed operations of the ORG signal, refer to section 4.3.3.

3.4 End-Limit Signals PEL and MEL

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

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

A circuit diagram is provided below. The external limit switch should have a contact capacity of +24V @ 6 mA minimum. Either 'A-type' (normal open) contact or 'B-type' (normal closed) contact switches can be used. To set the type of switch, configure dipswitch S1/SW2. By default, all bits of S1 on the card are set to ON (refer to section 2.10). For more details on EL operation, refer to section 4.3.2.

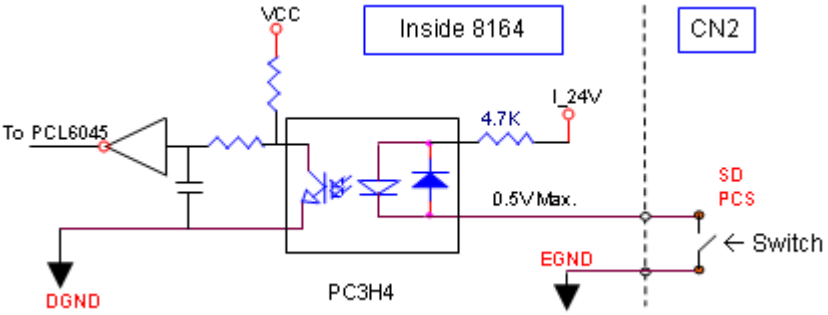


3.5 Ramping-down and PCS

There is a SD/PCS signal for each of the 4 axes. The signal names, pin numbers, and axis numbers are shown in the table below.

CN2 Pin No	Signal Name	Axis #
40	SD1/PCS1	1
46	SD2/PCS2	2
90	SD3/PCS3	3
96	SD4/PCS4	4

A circuit diagram is shown below. Typically, the limit switch is used to generate a slow-down signal to drive motors operating at slower speeds. For more details on SD/PCS operation, refer to section 4.3.1.

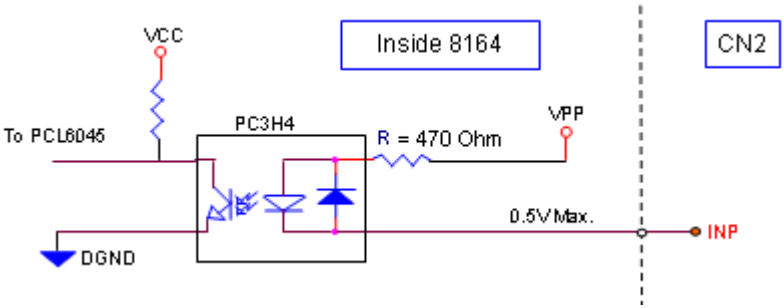


3.6 In-position Signal INP

The in-position signal INP from a servo motor driver indicates its deviation error. If there is no deviation error, then the servo's position indicates zero. The signal names, pin numbers, and axis numbers are shown in the table below.

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

The diagram below shows the input circuit of the INP signals.



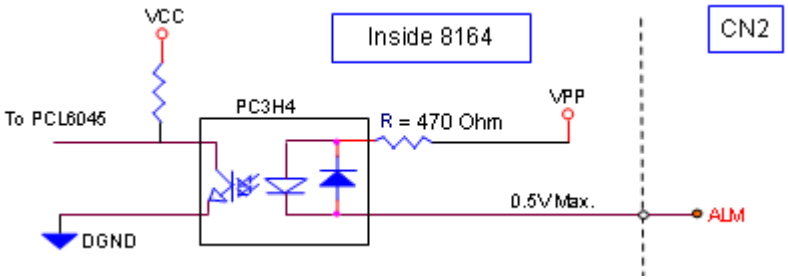
The in-position signal is usually generated by the servomotor driver and is ordinarily an open collector output signal. An external circuit must provide at least 5 mA current sink capabilities to drive the INP signal. For more details of INP signal operations, refer to section 4.2.1.

3.7 Alarm Signal ALM

The alarm signal ALM indicates the alarm status from the servo driver. The signal names, pin numbers, and axis numbers are shown in the table below.

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

The input alarm circuit diagram is provided. The ALM signal is usually generated by the servomotor driver and is ordinarily an open collector output signal. An external circuit must provide at least 5 mA current sink capabilities to drive the ALM signal. For more details of ALM signal operations, refer to section 4.2.2.



3.8 Deviation Counter Clear Signal ERC

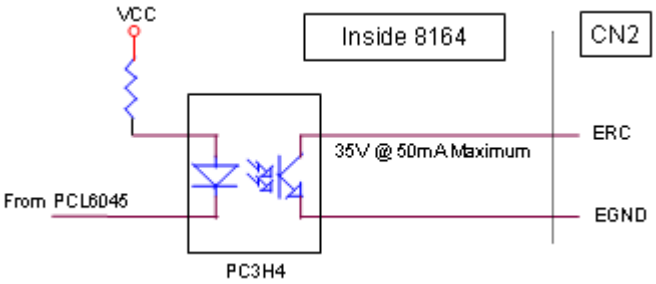
The deviation counter clear signal (ERC) is active for the following situations:

1. Home return is complete
2. 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 signal names, pin numbers, and axis numbers are shown in the table below.

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

The ERC signal clears the deviation counter of the servomotor driver. The ERC output circuit is an open collector with a maximum of 35V at 50 mA driving capacity. For more details on ERC operation, refer to section 4.2.3.

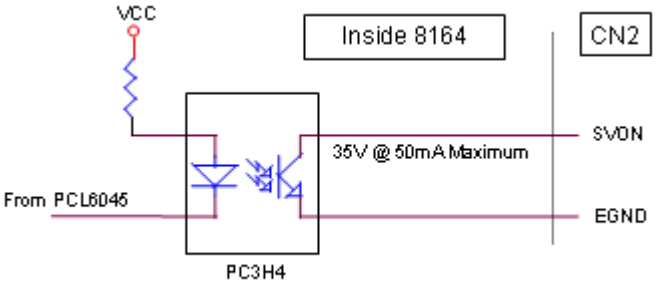


3.9 General-purpose Signal SVON

The SVON signal can be used as a servomotor-on control or general purpose output signal. The signal names, pin numbers, and its axis numbers 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 for the SVON signal is shown below:

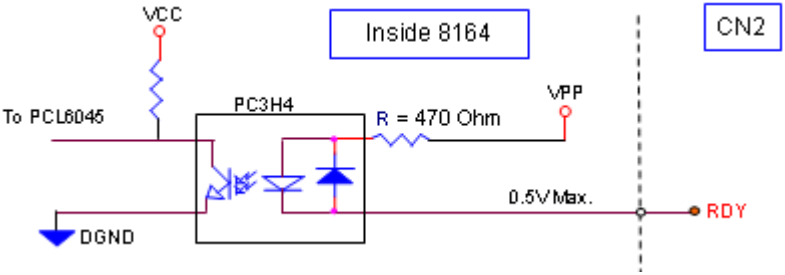


3.10 General-purpose Signal RDY

The RDY signals can be used as motor driver ready input or general purpose input signals. The signal names, pin numbers, and axis numbers are shown in the table below.

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 this diagram.



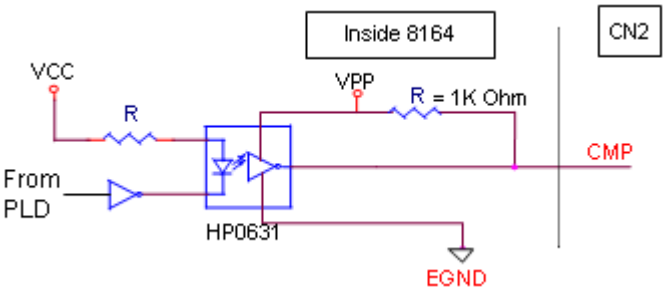
3.11 Position compare output pin: CMP

The card provides 2 comparison output channels: CMP1 and CMP2, used by the first 2 axes, 1 and 2. The comparison output channel generates a pulse signal when the encoder counter reaches a pre-set value set by the user.

The CMP channel is located on CN2. The signal names, pin numbers, and axis numbers are shown below.

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

The wiring diagram below shows the CMP on the first 2 axes.



NOTE CMP trigger type may be set to normal low (rising edge) or normal high (falling edge). Default setting is normal high. Refer to function _8164_set_trigger_type() in section 6.16 for details.

The CMP pin can be regarded as a TTL output.

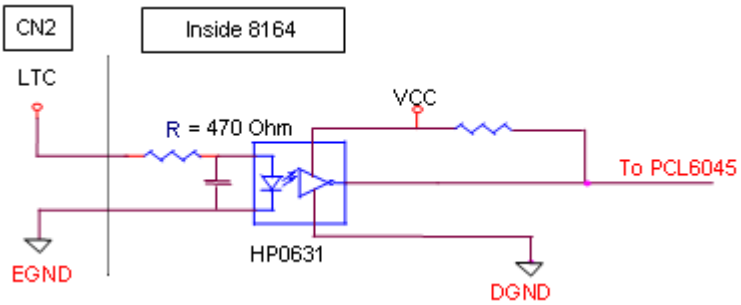
3.12 Position latch input pin: LTC

The card provides 2 position latch input channels: LTC3 and LTC4, used by the last 2 axes, 3 and 4. The LTC signal triggers the counter-value-capturing functions, which provides a precise position determination.

The LTC channel is on CN2. The signal names, pin numbers, and axis numbers are shown below.

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

The wiring diagram below shows the LTC of the last 2 axes.

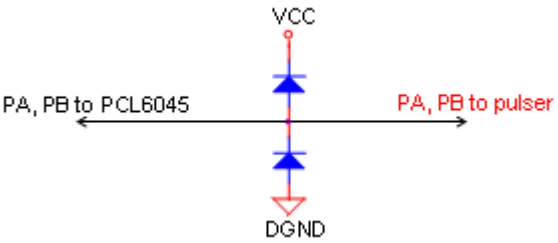


3.13 Pulser Input Signals PA and PB (PCI-8164 only)

The PCI-8164 accepts input pulser signals through the CN3 pins listed below. The pulses behave like an encoder. The signals generate the positioning information that guides the motor.

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

The CN3 PA and PB pins are directly connected to PA and PB pins of the PCL6045. The interface circuit is shown below.

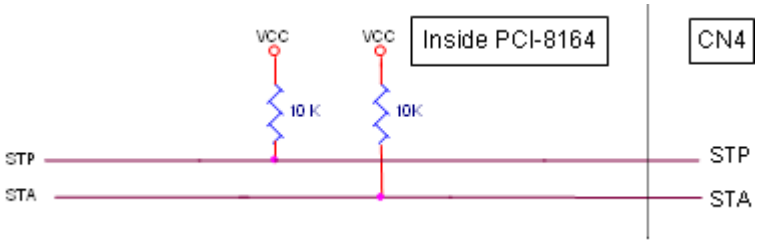


If the signal voltage of the pulser is not +5V or if the pulser is distantly placed, it is recommended that a photocoupler or a line driver be installed in between. Note that the CN3 +5V and DGND lines are provided from the PCI bus, and that this source is not isolated.

3.14 Simultaneously Start/Stop Signals STA and STP (PCI-8164 only)

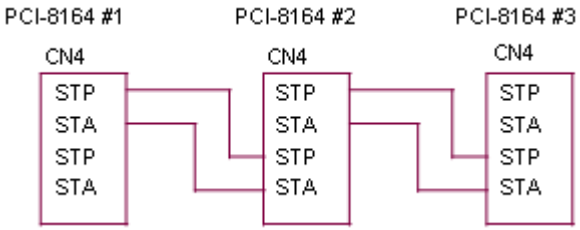
The PCI-8164 provides STA and STP signals that enable simultaneous start/stop of motions on multiple axes. The STA and STP signals are located on CN4.

The diagram below shows the tied STA and STP signals of the four axes.

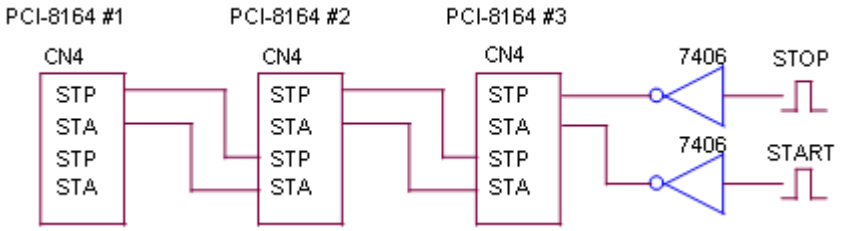


Both STP and STA signals are input and output signals. To operate the start and stop action simultaneously, both software control and external control are needed. With software control, the signals can be generated from any one of the PCL6045. You can also use an external open collector or switch to drive the STA/STP signals for simultaneous start/stop.

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



The following diagram shows how to allow an external signal to initiate the simultaneous start/stop connect a 7406 (open collector) or an equivalent circuit.

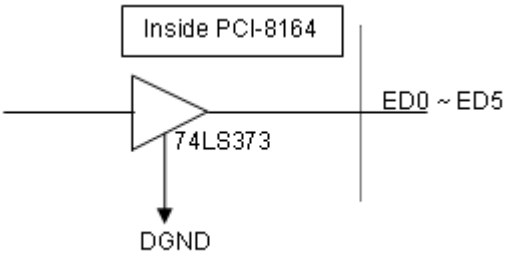

PCI-8164 #1
PCI-8164 #2
PCI-8164 #3
CN4
CN4
CN4
STP
STA
STP
STA
STP
STA
STP
STA
STP
STA
STP
STA
7406
7406
STOP
START

3.15 General Purpose TTL Output (PCI-8164 only)

The PCI-8164 provides six general purpose TTL digital outputs. The TTL output is located on CN5. The signal names, pin numbers, and axis numbers are listed below.

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 diagram shows the LTC of the last 2 axes.



3.16 Termination board

The card's CN2 can be connected with a DIN-100M15, including the ACL-102100 — a 100-pin SCSI-II cable. The DIN-100M15 is a general purpose 100-pin, SCSI-II DIN socket. It has convenient wiring screw terminals and an easy-install DIN socket that can be mounted to the DIN rails.

ADLINK also provides DIN-814M termination boards for Mitsubishi J2S servo motor drivers, DIN-814PA termination board for Panasonic Minas A servo motor drivers, DIN-814M-J3A termination board for Mitsubishi J3A Servo motor drviers, and DIN-814Y termination board for Yaskawa sigma-II servo motor driver.

3.17 General Purpose DIO (MPC-8164/PXI-8164 only)

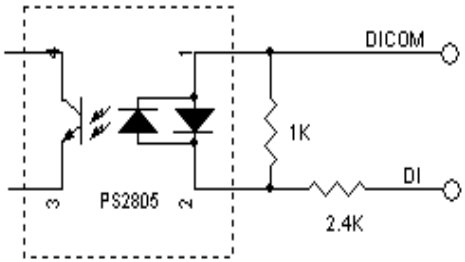
MPC-8164 has eight opto-isolated digital outputs and eight open collector digital inputs for general purpose use. Pin assignments are listed in the table below.

CN3 Pin No	Signal Name	CN3 Pin No	Signal Name
1	DOCOM	2	DOCOM
3	DOCOM	4	DOCOM
5	DO0	6	DO1
7	DO2	8	DO3
9	DO4	10	DO5
11	DO6	12	DO7
13	--	14	DICOM
15	DICOM	16	DICOM
17	DICOM	18	DI0
19	DI1	20	DI2
21	DI3	22	DI4
23	DI5	24	DI6
25	DI7	26	--

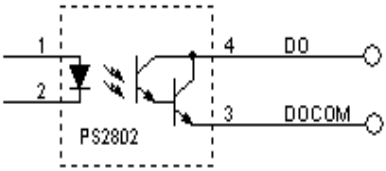
PXI-8164 has four opto-isolated digital outputs and four open collector digital inputs for general purpose use. Pin assignments are listed in the following table.

No.	Name	Function	No.	Name	Function
1	DICOM	Digital In Common	2	DOCOM	Digital Out Common
3	DI0	Input Channel 0	4	DO0	Output Channel 0
5	DI1	Input Channel 1	6	DO1	Output Channel 1
7	DICOM	Digital In Common	8	DOCOM	Digital Out Common
9	DI2	Input Channel 2	10	DO2	Output Channel 2
11	DI3	Input Channel 3	12	DO3	Ouput Channel 3
13	DICOM	Digital In Common	14	DOCOM	Digital Out Common
15	--	N/A	16	--	N/A
17	--	N/A	18	--	N/A
19	--	N/A	20	--	N/A

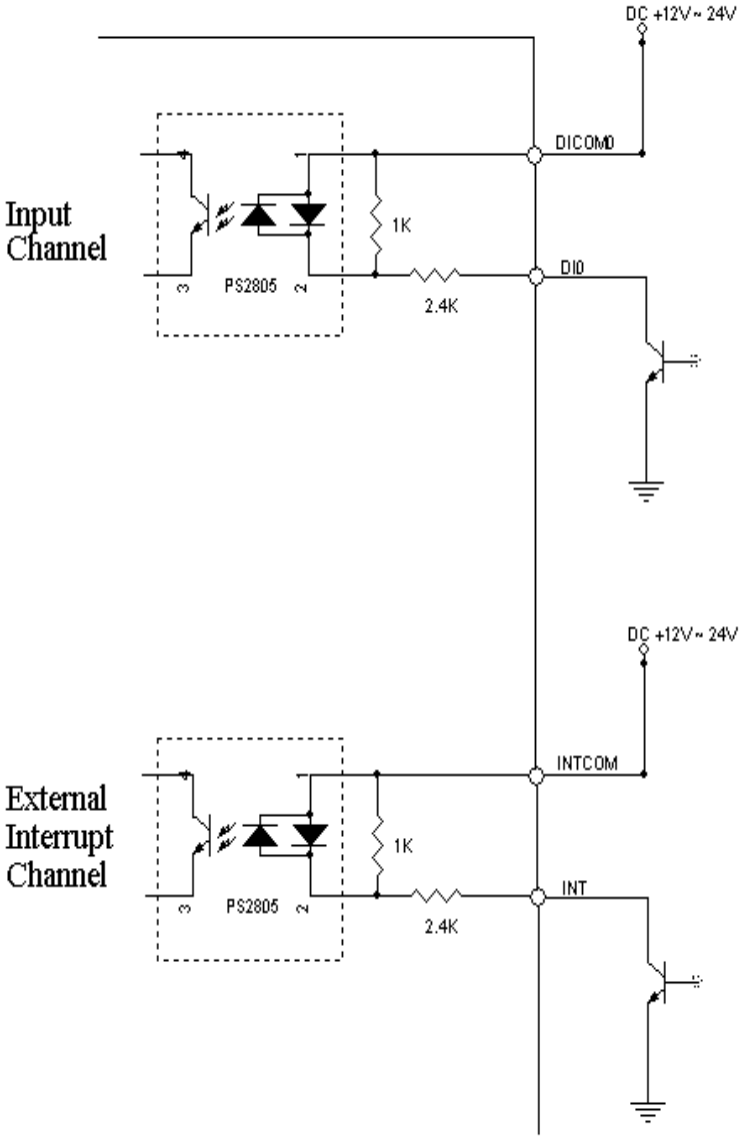
3.17.1 Isolated input channels



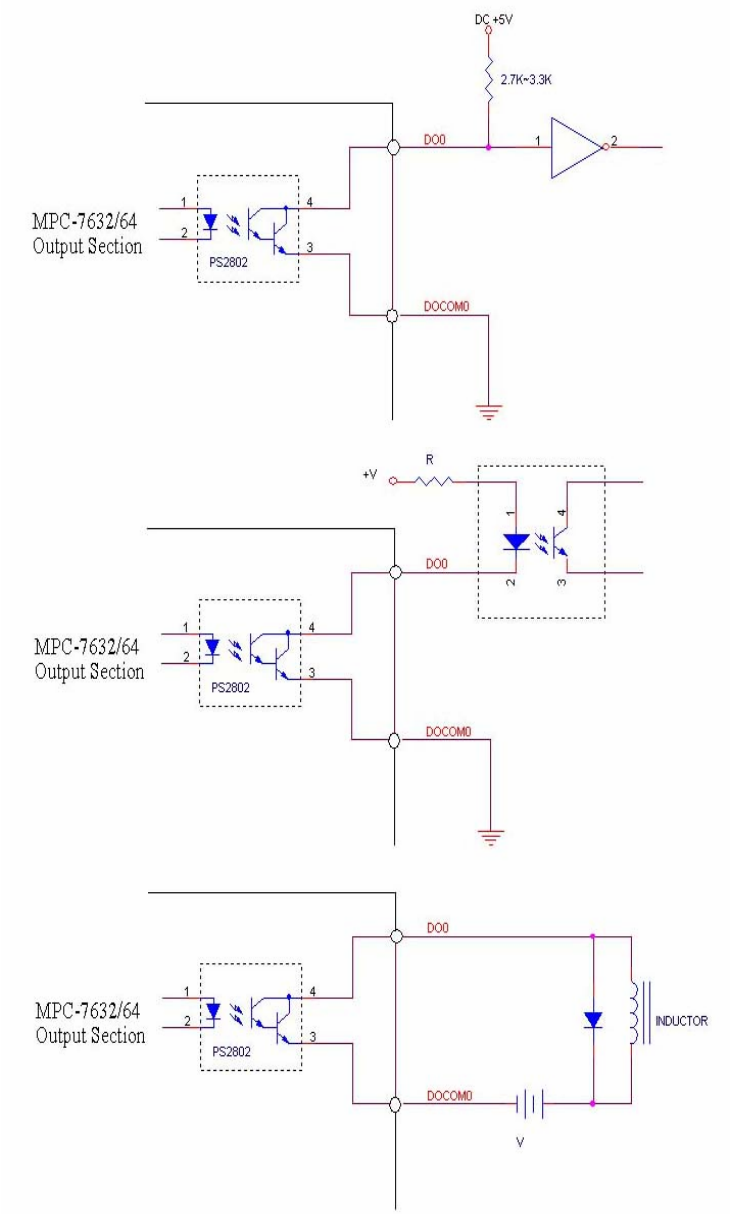
3.17.2 Isolated output channels



3.17.3 Example of input connection



3.17.4 Example of output connections



4 Operation Theory

This chapter describes the detailed operation of the 8164PCI-/MPC-/PXI-8164 card via the following sections:

- Section 4.1: The motion control modes
- Section 4.2: The motor driver interface (INP, ERC, ALM, SVON, RDY)
- Section 4.3: The limit switch interface and I/O status (SD/PCS, EL, ORG)
- Section 4.4: The counters (EA, EB, EZ)
- Section 4.5: Multiple card operation
- Section 4.6: Change position or speed on the fly
- Section 4.7: Position compare and latch
- Section 4.8: Hardware backlash compensator
- Section 4.9: Software limit function
- Section 4.10: Interrupt control
- Section 4.11: PXI Trigger Bus

4.1 Motion Control Modes

This section describes the pulse output signal configuration and motion control modes.

- 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 Circular interpolation with acc/dec time
- 4.1.8 Relationship between velocity and acceleration time
- 4.1.9 Continuous motion for multiple-axis
- 4.1.10 Home return mode for one axis
- 4.1.11 Home Search mode for one axis
- 4.1.12 Manual pulse mode for one axis
- 4.1.13 Synchronous starting modes

4.1.1 Pulse Command Output

The PCI-/MPC-/PXI-8164 uses pulse commands to control servo/stepper motors via the drivers. A pulse command has 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 mode. The modes vs. signal type of OUT and DIR pins are listed in the table below.

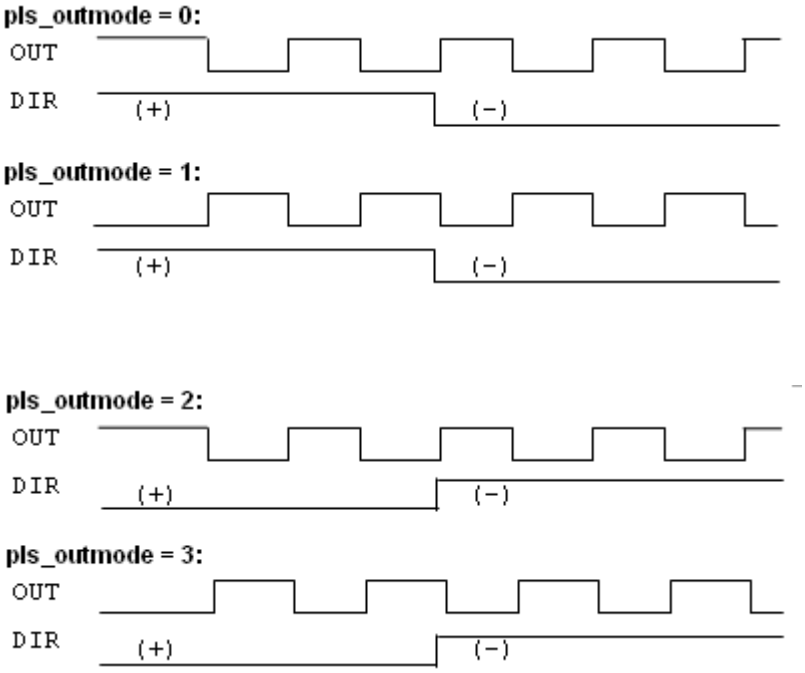
Mode	Output of OUT pin	Output of DIR pin
Dual pulse output (CW/CCW)	Pulse signal in plus (or CW) direction	Pulse signal in minus (or CCW) direction
Single pulse output (OUT/DIR)	Pulse signal	Direction signal (level)

The interface characteristics of these signals can be differential line driver or open collector output. Refer to section 3.1 for the jumper setting for different 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 positive (+) or negative (-). This mode is most commonly used. The diagrams below

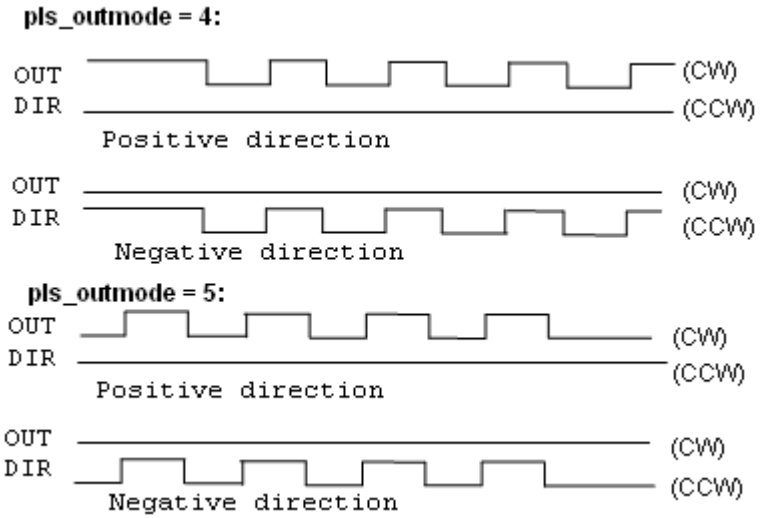
show the output waveform. It is possible to set the polarity of the pulse chain.



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 the CW pin makes the motor move in positive direction, whereas pulse output from the CCW pin makes the motor move in negative direction. The following dia-

gram shows the output waveform of positive (+) commands and negative (-) commands.

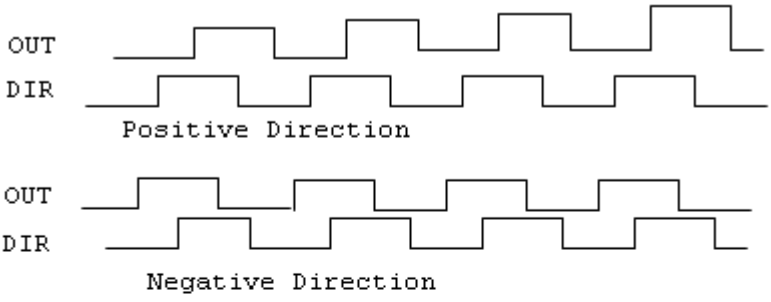


A/B Phase Pulse Output Mode (A/B phase Mode)

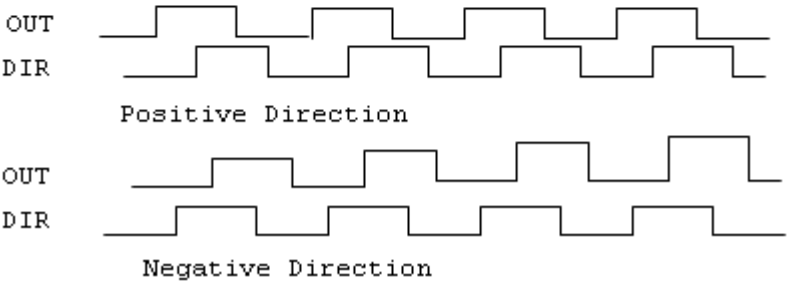
In this mode, the waveform of the OUT and DIR pins represent A-phase and B-phase pulse output, respectively. Pulses output from the OUT pin leading makes the motor move in positive direction, whereas pulse output from the DIR pin leading makes the motor move in negative direction. The following diagram shows the out-

put waveform of positive (+) commands and negative (-) commands. This mode is not available in older version boards.

pls_outmode = 5:



pls_outmode = 6:



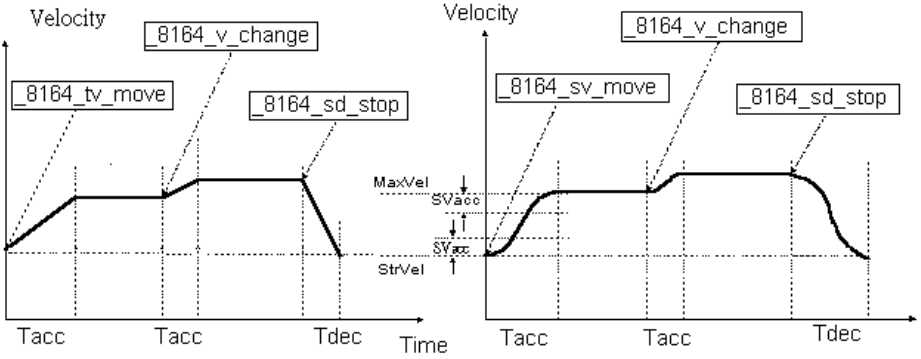
Related function:

- _8164_set_pls_outmode(): Refer to section 6.4.

4.1.2 Velocity mode motion

This mode is used to operate a one-axis motor with velocity mode motion. The output pulse accelerates from a starting velocity (StrVel) to a specified maximum velocity (MaxVel). The **`_8164_tv_move()`** function is used for constant linear acceleration while the **`_8164_sv_move()`** function is use for acceleration according to the S-curve. The pulse output rate is kept at maximum velocity until another velocity command is set or a stop command is issued. The **`_8164_v_change()`** is used to change the speed during an operation. Before this function is applied, make sure to call **`_8164_fix_speed_range()`**. Refer to section 4.6 for more information. The **`_8164_sd_stop()`** function is used to decelerate the motion until it stops. The **`_8164_emg_stop()`** function is used to immediately stop the motion. These change or stop functions follow the same velocity profile as its original move functions, **`tv_move`** or **`sv_move`**. The velocity profile is shown below.

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).



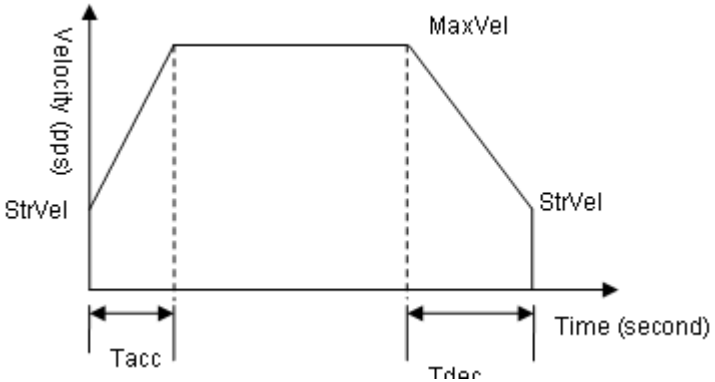
Related functions:

- _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 moves a singe axis motor to a specified position (or distance) with a trapezoidal velocity profile. The 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 cases, the acceleration and deceleration may be different. The function **`_8164_motion_done()`** is used to check whether the movement is completed.

The diagram shows the trapezoidal profile.



The card supports 2 trapezoidal point-to-point functions. In the **`_8164_start_ta_move()`** function, the absolute target position must be given in units of pulses. The physical length or angle of one movement is dependent on the motor driver and mechanism (including 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. The ratio between command pulses and external feedback pulse input must be appropriately set by the **`_8164_set_move_ratio()`** function.

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

The StrVel and MaxVel parameters are given in units of pulses per second (PPS). The Tacc and Tdec parameters are in units of second to represent accel./decel. time respectively. You must 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 units of pps/sec^2. The area inside the trapezoidal profile represents the moving distance.

Units of velocity setting are pulses per second (PPS). Usually, units of velocity of the manual of motor or driver are in rounds per minute (RPM). A simple conversion is necessary to match between these two units. Here we use an example to illustrate the conversion:

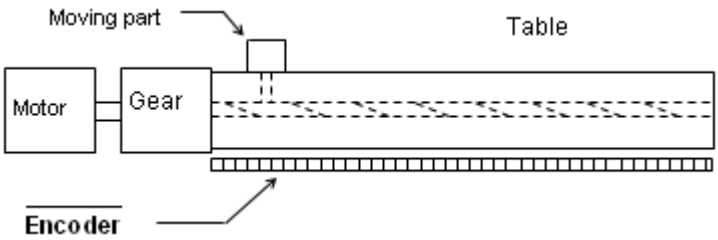
A servomotor with an AB phase encoder is used in a X-Y table. The resolution of encoder is 2000 counts per phase. The maximum rotating speed of motor is designed to be 3600 RPM. What is the maximum pulse command output frequency that you have to set on 8164?

Answer: MaxVel = 3600/60*2000*4 = 480000 PPS

Multiplying by 4 is necessary because there are four states per AB phase (See 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 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 the motor into linear motion (see the following diagram). If the resolution of the motor is 8000 pulses/round, then the resolution of the gear mechanism is 100 mm/round (i.e., part moves 100 mm if the motor turns one round). Then, the resolution of the command pulse will be 80 pulses/mm. If the resolution of the encoder mounting on the table is 200 pulses/mm, then you have to set the move

ratio to 200/80=2.5 using 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 8164 won’t recognize the actual absolute position during motion.

Related 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 moves a single-axis motor to a specified position (or distance) with an S-curve velocity profile. S-curve acceleration profiles are useful for both stepper and servomotors. The smooth transitions between the start of the acceleration ramp and transition to constant velocity produce less wear and tear than a trapezoidal profile motion. The smoother performance increases the life of the motor and the mechanics of the system.

There are several parameters that need to be set in order to make a S-curve move. These include:

Pos: target position in absolute mode, in units of pulses

Dist: moving distance in relative mode, in units of pulses

StrVel: start velocity, in units of PPS

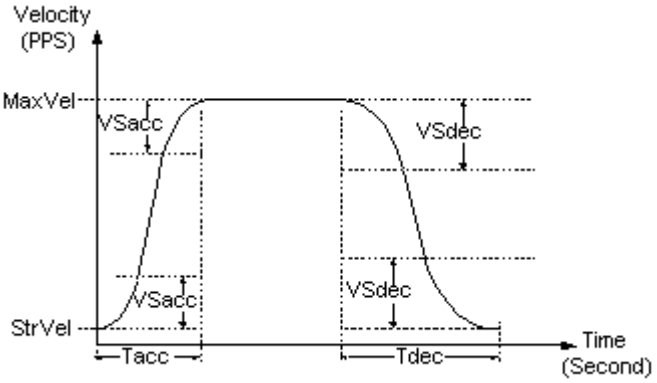
MaxVel: maximum velocity, in units of PPS

Tacc: time for acceleration (StrVel -> MaxVel), in units of seconds

Tdec: time for deceleration (MaxVel -> StrVel), in units of seconds

VSacc: S-curve region during acceleration, in units of PPS

VSdec: S-curve region during deceleration, in units of PPS



Normally, the accel/decel period consists of three regions: two VSacc/VSdec curves and one linear. During 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 + VSacc). 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 is similar in fashion.

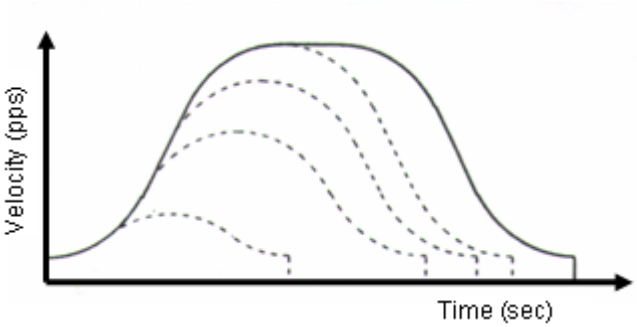
Special case:

*If you want to disable the linear region, the VSacc/VSdec must be assigned **0** rather than **0.5** (MaxVel-StrVel).*

Remember that the VSacc/VSdec is in units of PPS and it should always keep in the range of [0 to (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), then the maximum velocity is automatically lowered (see the figure below).

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



Related 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 differences between all single axis motion functions, including preset mode (both trapezoidal and S-curve motion) and constant velocity mode.

	Velocity Profile			
	Trapezoidal	S-Curve	Relative	Absolute
_8164_tv_move	✓		-------	-------
_8164_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		✓		✓

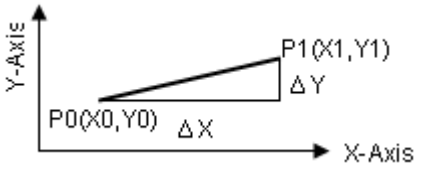
4.1.5 Linear interpolation for 2-4 axes

In this mode, any two of four, three of four, or all four 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.

Note that you cannot use two groups of two axes for linear interpolation on a single card at the same time. You can however, use one 2-axis linear and one 2-axis circular interpolation at the same time. If you want to stop an interpolation group, use the **`_8164_sd_stop()`** or **`_8164_emg_stop()`** function.

2 axes linear interpolation

In the diagram below, 2-axis linear interpolation means to move the XY position (or any two of the four axis) 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:

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

When calling 2-axis linear interpolation functions, the **vector speed** needs to define the start velocity, **StrVel**, and maximum velocity, **MaxVel**. Both trapezoidal and S-curve profiles are available.

Example:

_8164_start_tr_move_xy(0, 30000.0, 40000.0, **1000.0**, **5000.0**, 0.1, 0.2) will cause the XY axes (axes 0 & 1) of Card 0 to perform a linear interpolation movement, in which:

ΔX = 30000 pulses; ΔY = 40000 pulses

Start vector speed=1000pps, X speed=600pps, Y speed = 800pps

Max. vector speed =5000pps, X speed=3000pps, Y speed = 4000pps

Acceleration time = 0.1sec; Deceleration time = 0.2sec

There are two groups of functions that provide 2-axis linear interpolation. The first group divides the four 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 you to freely assign the two 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”, and “a” after _8164_start mean:

t – Trapezoidal profile

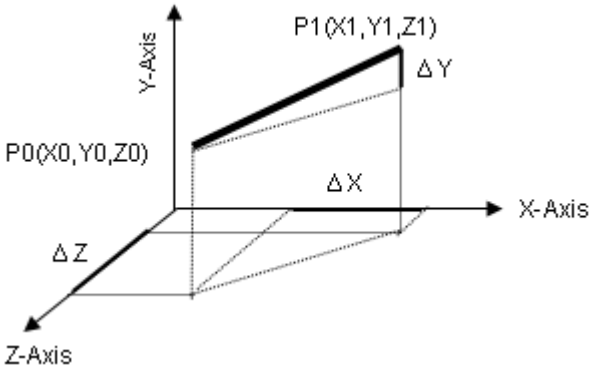
s – S-Curve profile

r – Relative motion

a – Absolute motion

3-axis linear interpolation

Any three of the four axes of the card may perform 3-axis linear interpolation. As shown the figure below, 3-axis 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, starting and stopping 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 3-axis linear interpolation functions, the vector speed is needed to define the start velocity, **StrVel**, and maximum velocity, **MaxVel**. Both trapezoidal and S-curve profiles are available.

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 speed = 100/$\sqrt{14}$ = 26.7pps
- Y speed = 2*100/$\sqrt{14}$ = 53.3pps
- Z speed = 3*100/$\sqrt{14}$ = 80.1pps

Max. vector speed =5000pps,

- X speed= 5000/$\sqrt{14}$ = 1336pps
- Y speed = 2*5000/$\sqrt{14}$ = 2672pps
- Z speed = 3*5000/$\sqrt{14}$ = 4008pps

The following functions are used for 3-axis linear interpolation: *_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”, and “a” after _8164_start mean:

t – Trapezoidal profile

s – S-Curve profile

r – Relative motion

a – Absolute motion

4-axis linear interpolation

With 4-axis 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 following functions are used for 4-axis linear interpolation:

_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”, and “a” after _8164_start mean:

t – Trapezoidal profile

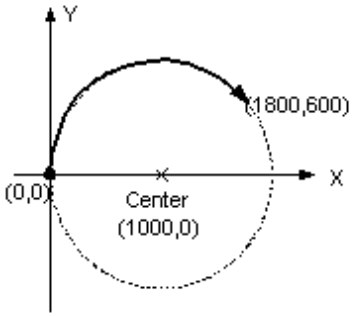
s – S-Curve profile

r – Relative motion

a – Absolute motion

4.1.6 Circular interpolation for 2 axes

Any 2 of the 4 axes of the card can perform circular interpolation. In the example below, 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 tangential speed. Notice that if the end point of arc is not at a proper position, it will move circularly without stopping.



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 is stopped when either of the two axes reached end point.

There are two groups of functions that provide 2-axis circular interpolation. The first group divides the four axes into XY (axis 0 and

axis 1) and ZU (axis 2 and 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)

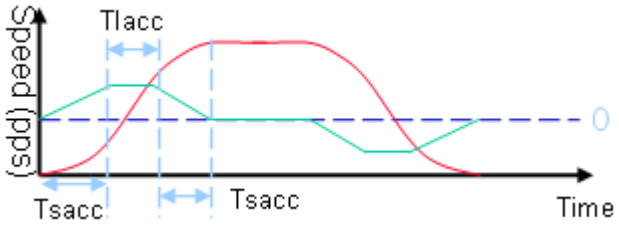
The second group allows user to freely assign any targeted two axes.

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

4.1.7 Circular interpolation with Acc/Dec time

In section 4.1.6, the circular interpolation functions do not support acceleration and deceleration parameters. Therefore, they cannot perform a T or S curve speed profile during operation. However, sometimes the need for an Acc/Dec time speed profile will help a machine to make more accurate circular interpolation. The 8164 card has another group of circular interpolation functions to perform this type of interpolation, but requires the use of Axis3 as an aided axis, which means that Axis3 cannot be used for other purposes while running these functions. For example, to perform a circular interpolation with a T-curve speed profile, the function **`_8164_start_tr_arc_xyu()`** is used. This function will use Axis0 and Axis1, and also Axis3 (Axis0=x, Axis1=y, Axis2=z, Axis3=u). For the full lists of functions, refer to section 6.8.

To check if the card supports these functions use the **`_8164_version_info()`** function. If hardware information for the card returns a value with the 4th digit greater then 0, for example '1003', users can use this group of circular interpolation to perform S or T-curve speed profiles. If the hardware version returns a value with the 4th digit being 0, then the card does not support these functions.

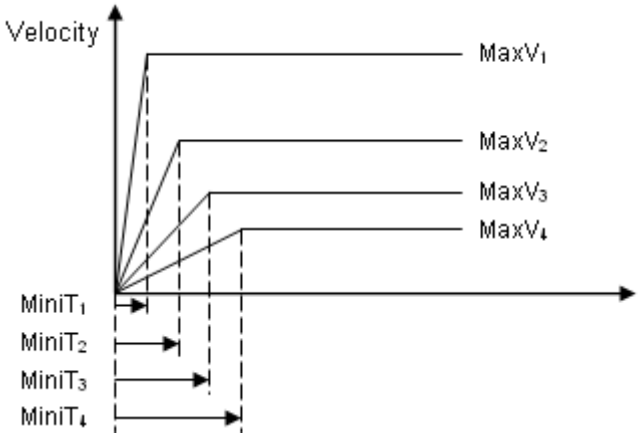


4.1.8 Relationship between velocity and acceleration time

The maximum velocity parameter of a motion function will eventually have a minimum acceleration value. This means that there is a range for acceleration time over one velocity value. Under this relationship, to obtain a small acceleration time, a higher maximum velocity value to match the smaller acceleration time is required. Function **`_8164_fix_speed_range()`** will provide such operation. This function will raise the maximum velocity value, which in turn results in a smaller acceleration time. Note it does not affect the actual end velocity. For example, to have a 1ms acceleration time from a velocity of 0 to 5000 (pps), the function can be inserted before the motion function as shown.

_8164_fix_speed_range(AxisNo,OverVelocity);

_8164_start_tr_move(AxisNo,5000,0,5000,0.001,0.001);



How do users decide an optimum value for “OverVelocity” in the _8164_fix_speed_range() function? The _8164_verify_speed() function is provided to calculate such value. The inputs to this function are the start velocity, maximum velocity and over velocity values. The output value will be the minimum and maximum values of the acceleration time.

For example, if the original acceleration range for the command is:

_8164_start_tr_move(AxisNo,5000,0,5000,0.001,0.001),

then use the following function:

_8164_verify_speed(0,5000,&minAccT, &maxAccT,5000);

The value miniAccT will be 0.0267sec and maxAccT will be 873.587sec. This minimum acceleration time does not meet the requirement of 1mS. To achieve such a low acceleration time the over speed value must be used.

By changing the OverVelocity value to 140000,

_8164_verify_speed(0,5000,&minAccT, &maxAccT,140000);

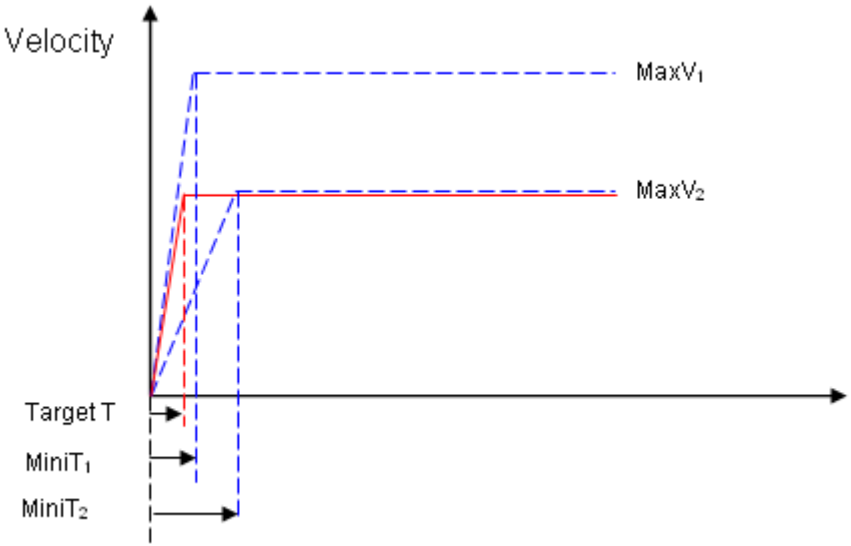
The value miniAccT will be 0.000948sec and maxAccT will be 31.08sec. This minimum acceleration time meets the requirements. So, the motion command can be changed to:

_8164_fix_speed_range(AxisNo,140000);

_8164_start_tr_move(AxisNo,5000,0,5000,0.001,0.001);

Notes:

- The return value of **`_8164_verify_speed()`** is the minimum velocity of motion command, it does not always equal to your start velocity setting. In the above example, it will be 3pps more than the 0pps setting.
- To disable the fix speed function **`_8164_fix_speed_range()`** use **`_8164_unfix_speed_range()`**
- Minimize the use of the OverVelocity operation. the more it is used, the coarser the speed interval is.



Example:

User's Desired Profile: (MaxV2, Target T) is not possible under MaxV2 according to the (MaxV, MiniT) relationship. So one must change the (MaxV, MiniT) relationship to a higher value, (MaxV1, MiniT1). Finally, the command would be:

_8164_fix_speed_range(AxisNo, MaxV1);

_8164_start_tr_move(AxisNo,Distance, 0 , MaxV2 , Target T, Target T);

Related functions:

- _8164_fix_speed_range(), _8164_unfix_speed_range(),_8164_verify_speed()

Refer to section 6.5

4.1.9 Continuous motion

The card allows you 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 a user calls the following function to perform a single axis preset motion:

1) _8164_set_continuous_move(0, 1)

It enables the continuous move function by keeping current position in internal variable. If this function is not enabled, the second motion function will return busy status and can not do continuous motion.

2) _8164_start_ta_move(0, 50000, 100, 30000, 0.1, 0.0)

It causes the axis “0” to move to position “50000.0.” Before the axis arrives, the user can call a second motion (refer to the next function). Notice that the deceleratin of this function is set to 0. It means that deceleration is not needed in this command in order to smoothly link the next command velocity.

3) _8164_start_tr_move(0, 20000, 100, 30000, 0.0, 0.2)

The second function call does not affect the first one. Actually, it will be executed and written into the card pre-register. After the first move is finished, the card will continue with the second move according to the pre-registered value. The time interval between these two moves can be seen as a continuous move and pulses will be continuously be generated at the “50000.0” position. Notice that the acceleration time is set to 0. It means that we do not need acceleration in this command in order to smoothly link the previous command velocity.

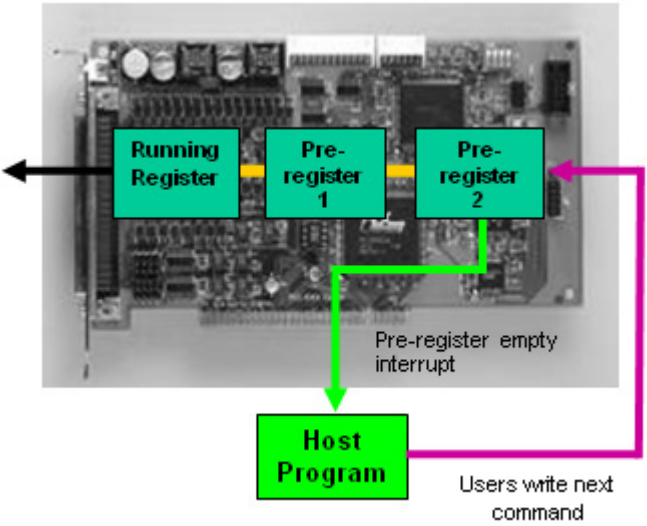
4) _8164_set_continuous_move(0, 0)

Return to normal move mode.

The theory of continuous motion is described below:

Theory of continuous motion (FIFO architecture)

The following diagram shows the register data flow of the card.



1. The first motion is executed and the CPU writes corresponding values into Pre-Register2.

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

2. Since Pre-Register1 and Register are empty, the data in pre-register 2 is automatically moved to the Register and executed immediately by the ASIC.
3. The second function is called. The CPU writes the corresponding values into pre-register2.

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

4. Since Pre-register1 is empty; the data in pre-register 2 is automatically moved to Pre-Rregister1 and waits to be executed.
5. You may can execute a 3rd function, and it will be stored to Pre-register2.
6. When the first function is completed, the Register becomes empty, and data in pre-register1 is allowed to move to Register and is executed immediately by the

ASIC. Data in Pre-Register2 is then moved to Pre-Register1.

7. The ASIC will inform the CPU generating an interrupt that a motion is completed. Users can then write the 4th motion command into Pre-Register2.

Procedures for continuous motion (by interrupt)

The following procedures are to help user making continuous motion.

1. (If under DOS): Enable the interrupt service using **`_8164_int_contol()`**

   (If under Windows): Enable the interrupt service using **`_8164_int_contol()`** and **`_8164_int_enable()`**.
2. Set bit "2" of INT factor to be "True" using **`_8164_set_int_factor()`**.
3. Set the "conti_logic" to be "1" by: **`_8164_set_continuous_move()`**
4. Call the first three motion functions.
5. Wait for interrupt in ISR (under DOS) or Event (under Windows) of 2nd pre-register empty.
6. Call the fourth motion function.
7. Wait for interrupt in ISR (under DOS) or Event (under Windows) of 2nd pre-register empty.
8. Repeat steps 6 and 7 until all motion functions are called.

Procedures for continuous motion (by polling)

Another method to determine a motion-completed action is by polling buffer empty status. User may constantly check the buffer status using the **`_8164_check_continuous_buffer()`** function before calling a new motion function.

Restrictions of continuous motion

Here are restrictions and suggestions for continuous motion:

1. When the Pre-Registers are full, you may not execute any more motion functions. Otherwise, the new function one will overwrite the existing function in Pre-Register2.
2. To get a continuity of velocity between two motions, the previous end velocity of and starting velocity of the next must be the same. There are several methods to achieve this. The easiest way is to set the deceleration/ acceleration time to **0**.
3. The dimension of axes can not be changed during continuous motion

For example:

_8164_set_continuous_move(0, 1)

_8164_set_continuous_move(1, 1)

_8164_set_inp(0,0,0)

_8164_set_inp(1,0,0)

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

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

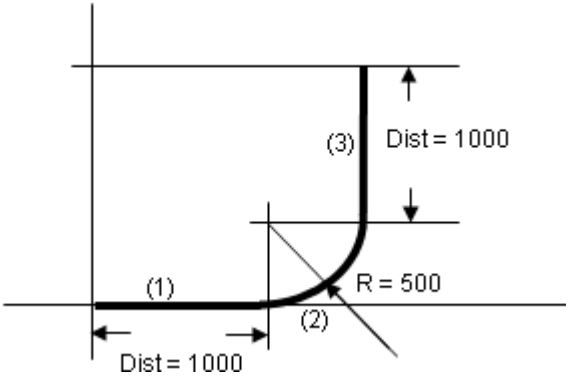
_8164_set_inp(0,1,0)

_8164_set_inp(0,1,0)

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

_8164_set_continuous_move(0, 0)

_8164_set_continuous_move(1, 0)



Explanation of example:

When these three motions ARE executed sequentially, the 1st occupies the Register and is executed immediately; the 2nd occupies Pre-Register1 and waits for completion of the 1st motion. The 3rd occupies Pre-Register2 and waits for completion of the 2nd motion. Since the 1st motion has a **0** deceleration time and the 2nd motion is an arc of constant velocity, which is the same as the max_vel of the 1st, the 8164 will output a constant frequency at intersections between them.

1. Continuous motion between different axes is meaningless. Different axes have their own register and pre-register system.
2. Continuous motion between different numbers of axes is not allowed. For example: **`_8164_start_tr_move()`** can not be followed by **`_8164_start_ta_move_xy()`** nor vice versa.
3. It is possible to perform a 3-axis or 4-axis continuous linear interpolation, but speed continuity is impossible to achieve.
4. If any absolute mode is used during continuous motion, make sure that **`_8164_reset_target_pos()`** is exe-

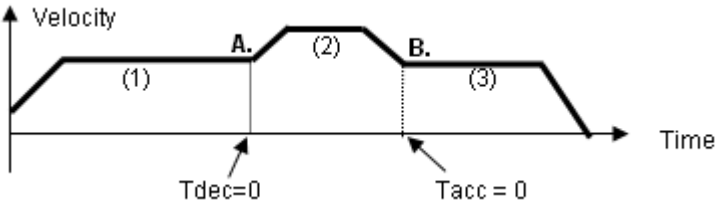
cuted at least once after home move. Refer to 4.1.8: Home return mode for more details.

5. INP enable could be set before any motion command. In continuous motion, normally you disable INP except final command, since you want velocity command to be continuously sent without slowing down by INP between every motion.

Examples of continuous motion

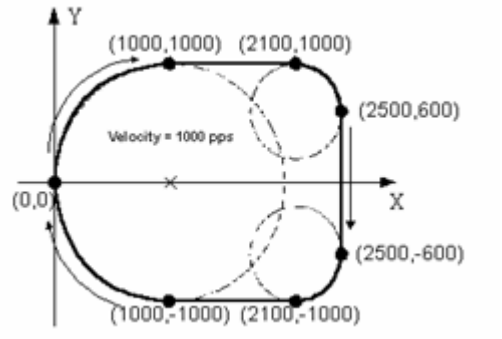
The following are examples of continuous motion:

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



This example demonstrates how to use the continuous motion function to achieve velocity changing at pre-set points. The 1st motion (ta) moves the axis to point A, with Tdec =0, and then the 2nd continues immediately. 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 continued.

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. Refer to the previous example.

The functions related to continuous motion are listed below:

- _8164_set_continuous_move(), _8164_check_continuous_buffer()

Refer to section 6.17.

4.1.10 Home Return Mode

In this mode, the card is allowed to continuously 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 the 8164. The **home_mode** of function **`_8164_set_home_config()`** is used to select whichever mode is preferred.

After completion of home move, it is necessary to keep in mind that all related position information should be reset to be **0**. The card has 4 counters and 1 software-maintained position recorder, including:

- Command position counter: counts the number of pulse outputs
- Feedback position counter: counts the number of pulse inputs
- Position error counter: counts the error between command and feedback pulse numbers.
- General-Purposed counter: can be configured as pulse output, feedback pulse, manual pulse, or CLK/2.
- Target position recorder: records the target position. (Software mainained)

Refer to section 4.4 for details on the position counters.

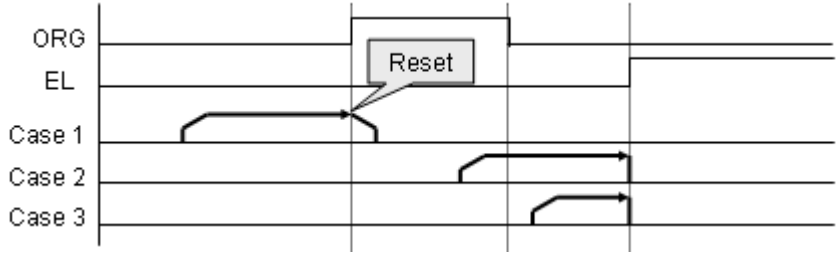
After home move is complete, the first four counters will be automatically cleared to 0, however, the target position recorder will not. Because it is software maintained, it is necessary to manually set the target position to **0** by calling the function _8164_reset_target_pos().

In some home mode, the stop position after homing is not at 0 because of deceleration. In this case, users must call _8164_reset_target_pos() by entering this position.

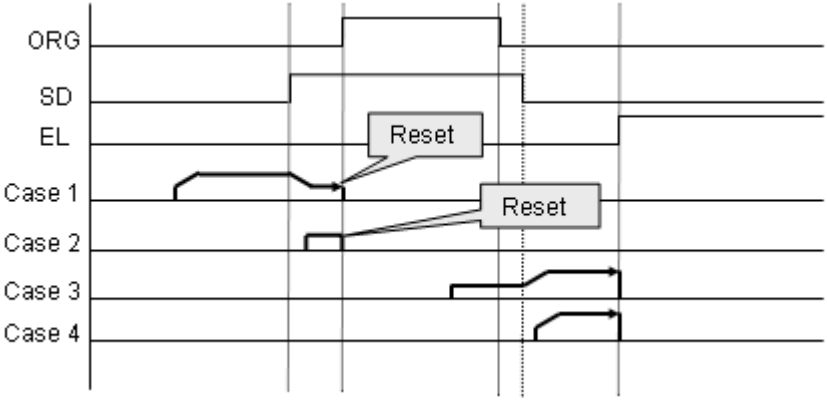
The following figures show the various home modes:

home_mode=0: ORG > Slow down > Stop

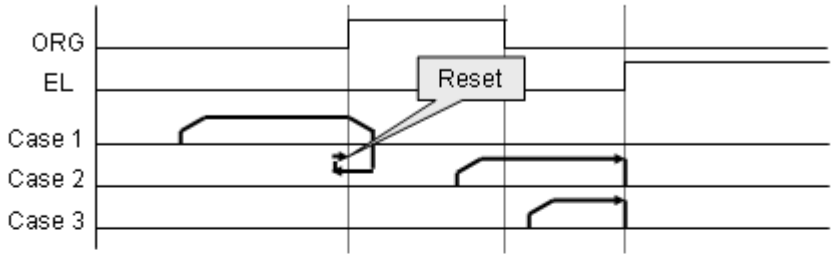
- When SD (Ramp-down signal) is inactive.



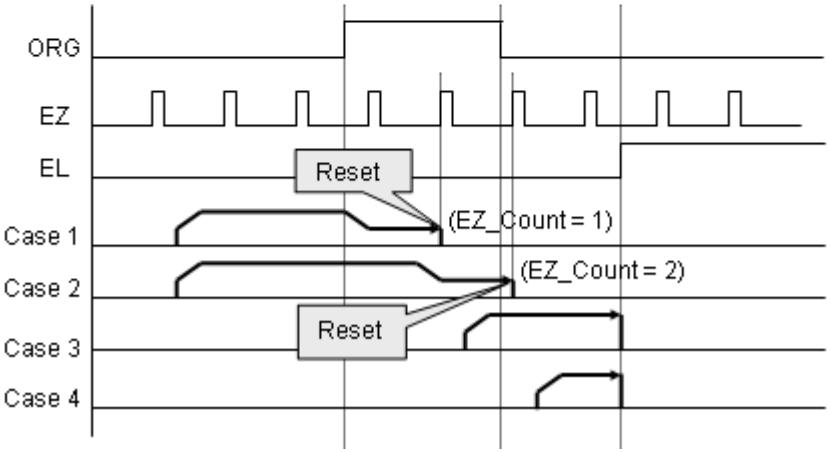
- When SD (Ramp-down signal) is active.



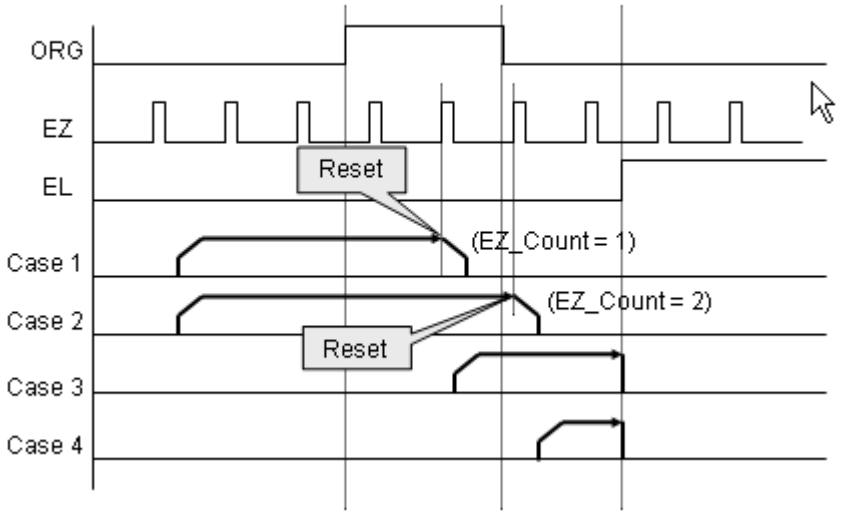
home_mode=1: ORG → Slow down → Stop on ORG edge



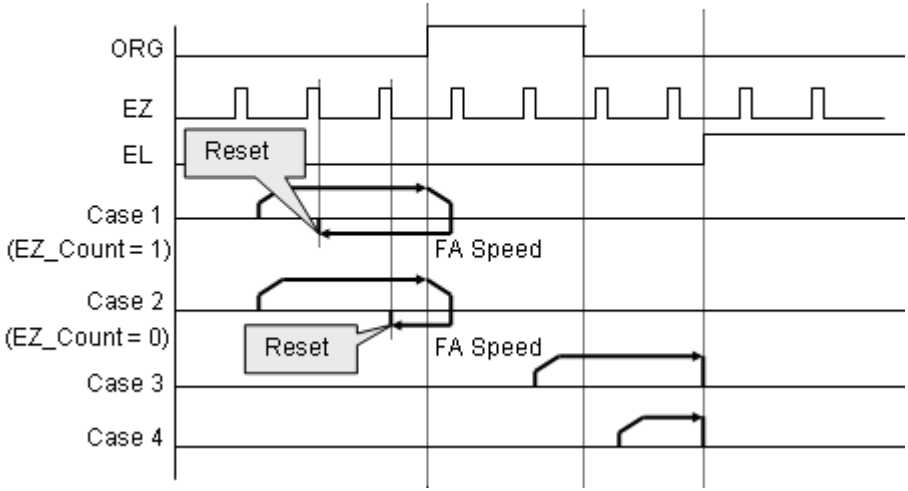
home_mode=2: ORG → Slow down → Stop on EZ signal



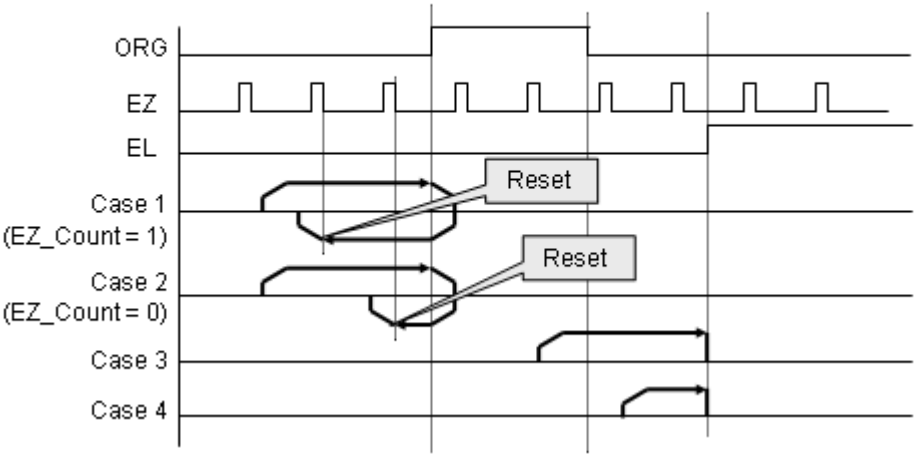
home_mode=3: ORG → EZ → Slow down → Stop



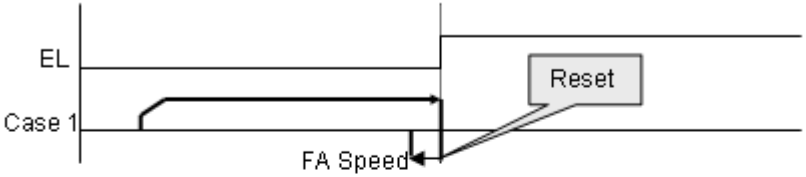
home_mode=4: ORG → Slow down → Go back at FA speed → EZ → Stop



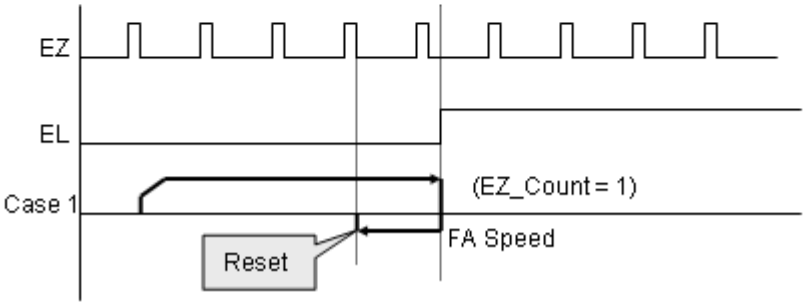
home_mode=5: ORG → Slow down → Go back → Accelerate to MaxVel → EZ → Slow down → Stop



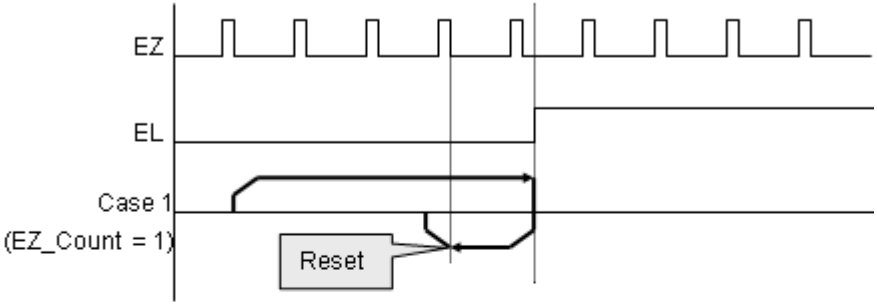
home_mode=6: *EL only*



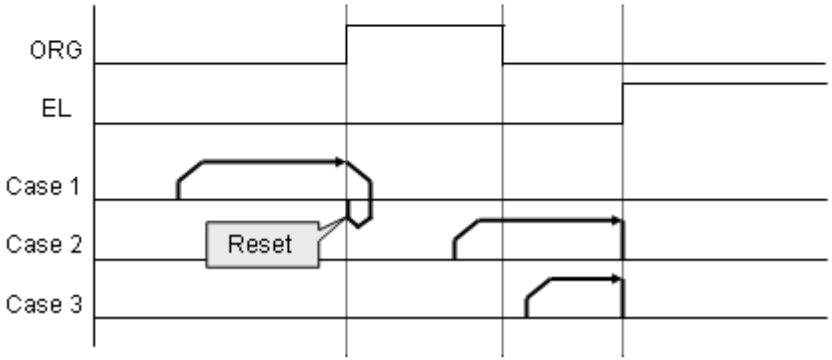
home_mode=7: EL → Go back → Stop on EZ signal



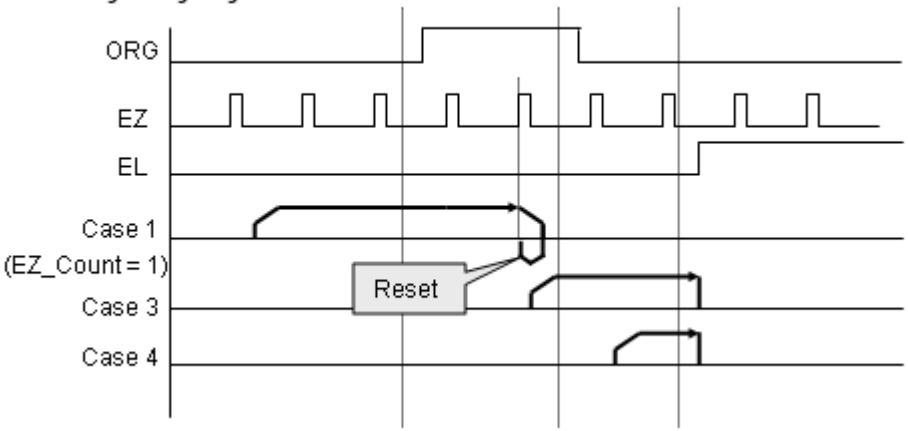
home_mode=8: EL → Go back → Accelerate to MaxVel →EZ → Slow down → Stop



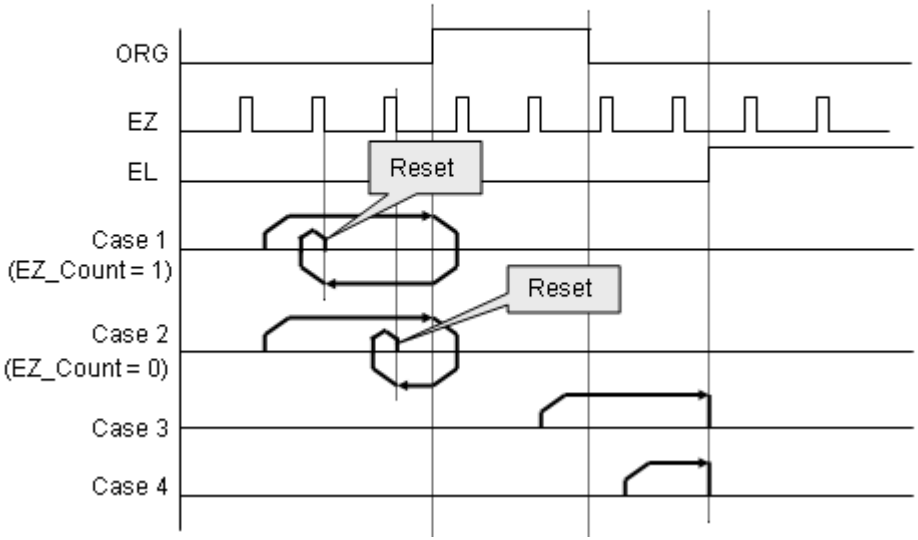
home_mode=9: ORG → Slow down → Go back → Stop at beginning edge of ORG



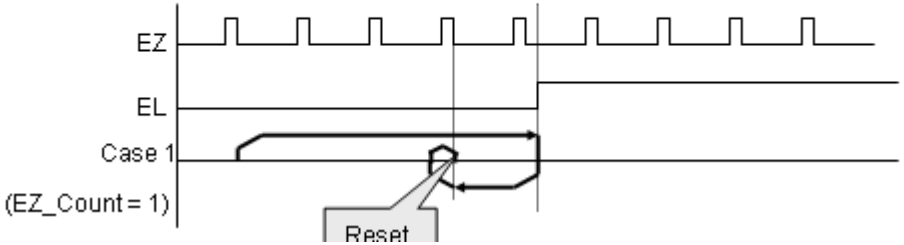
home_mode=10: ORG → EZ →Slow down → Go back → Stop at beginning edge of EZ



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



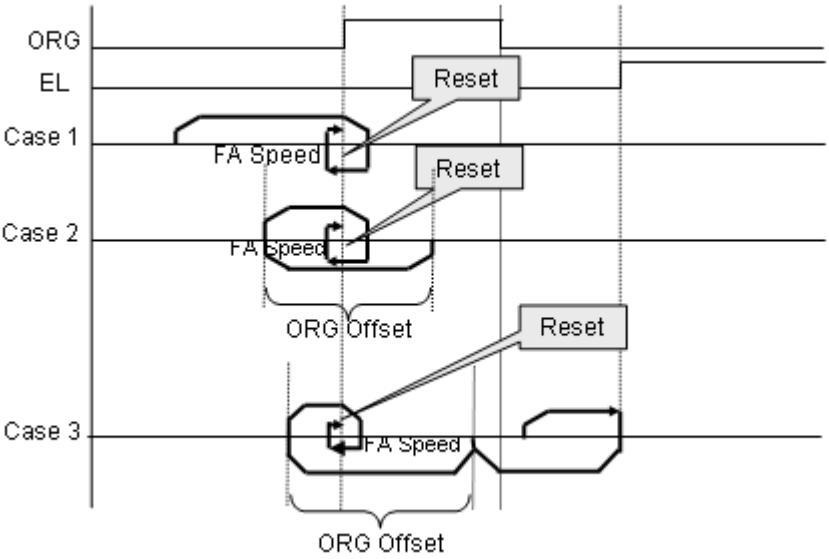
Relative Functions:

- _8164_set_home_config(), _8164_home_move(), _8164_set_fa_speed(), _8164_escape_home(), Refer to section 6.9.

4.1.11 Home Search Mode

This mode adds auto searching function on normal home return mode described in section 4.1.10. After **`_8164_home_config()`** is set, without calling **`_8164_home_move()`** but use **`_8164_home_search()`** to perform this function. The following illustration shows an example of home mode 2. The ORG offset must not be zero. Suggested value is the double length of ORG area:

.



Related functions:

- _8164_set_home_config(), _8164_home_move(), _8164_home_search(), _8164_set_fa_speed(), _8164_escape_home(), Refer to section 6.9.

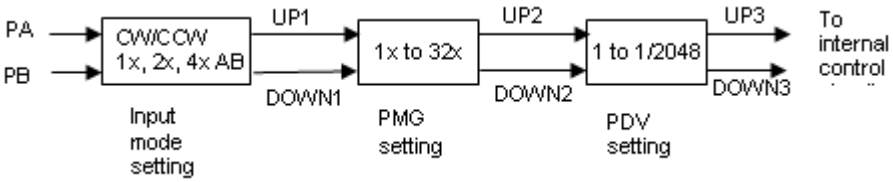
4.1.12 Manual Pulser Mode (PCI-8164 Only)

For manual operation of a device, you may use a manual pulse such as a rotary encoder. The PCI-8164 can receive input signals from a pulser and output its corresponding pulses from the OUT and DIR pins, thereby allowing a simplified external circuit.

This mode is effective when the **`_8164_pulser_vmove()`**, **`_8164_pulser_pmove()`**, or **`_8164_pulser_home_move()`** command has been called. To terminate the command use the **`_8164_sd_stop()` or `_8164_emg_stop()`** command.

The PCI-8164 receives positive and negative pulses (CW/CCW) or 90-degree phase difference signals (AB phase) from the pulser at the PA and PB pins. To set the input signal modes of the pulser, use the **`_8164_set_pulser_iptmode()`** function. The 90-degree phase difference in signals can be set by a multiplication of 1, 2, or 4. If the AB phase input mode is selected, PA and PB signals should have a 90-degree phase shift, and the position counter increases when the PA signal is leading the PB signal by 90 degrees.

The following figure shows pulser ratio block diagram.



Related functions:

- _8164_pulser_vmove(), _8164_pulser_pmove(), _8164_pulser_home_move(), _8164_set_pulser_iptmode(), _8164_set_pulser_ratio(), _8164_pulser_r_line2(), _8164_pulser_r_arc2(), _8164_pulser_a_line2(), _8164_a_arc2()

Refer to section 6.10

4.1.13 Synchronous starting modes

Synchronous motion means more than one axes can be started by a synchronous signal. The has three kinds of this motion.

1. Simultaneously start/stop by externl signal (STA/STP).

   Example: 3 axes move simultaneously

```
I16 AxisArray[3]={0,1,2};
I16 DistArray[3]={5000,6000,2000};
I16 StartV[3]={0,0,0};
I16 MaxV[3]={10000,20000,50000};
I16 Tacc[3]={0.02, 0.02, 0.02};
I16 Tdec[3]={0.03, 0.03, 0.03};
_8164_set_tr_move_all(3, AxisArray, DistArray,
     StartV, MaxV, Tacc,Tdec);
```

   It will wait STA signal to start. You can call **`_8164_stop_move_all(0)`** to cancel it.

2. Simultaneously start/stop by software function.

   Example: 3 axes move simultaneously

```
I16 AxisArray[3]={0,1,2};
I16 DistArray[3]={5000,6000,2000};
I16 StartV[3]={0,0,0};
I16 MaxV[3]={10000,20000,50000};
I16 Tacc[3]={0.02, 0.02, 0.02};
I16 Tdec[3]={0.03, 0.03, 0.03};
_8164_set_tr_move_all(3, AxisArray, DistArray,
     StartV, MaxV, Tacc,Tdec);
_8164_start_move_all(0);
```

3. Immediately start when other axes stop.

   Example: Axis1 starts after Axis0 stops

```
_8164_set_sync_option(1,0,0,3,1);
_8164_start_tr_move(1, 1000, 0, 5000, 0.01,
     0.01);
_8164_start_tr_move(0, 500, 0, 1000, 0.01, 0.01);
```

4. Immediately start when other axis' synchronous condition is satisfied.

Example: Axis2 starts when Axis1's acceleration ends.

```
_8164_set_sync_option(2,0,0,2, 0);
_8164_set_sync_signal_source(2, 1);
_8164_set_sync_signal_mode(1, 9);
_8164_start_tr_move(1, 1000, 0, 5000, 0.01,
     0.01);
_8164_start_tr_move(0, 500, 0, 1000, 0.01, 0.01);
```

Related functions:

- _8164_set_sync_signal_mod(), _8164_set_sync_signal_source(), _8164_set_sync_option(), _8164_set_tr_move_all(), _8164_start_move_all(), Refer to section 6.18

4.2 The motor driver interface

The card provides the INP, ALM, ERC, SVON, and RDY signals for a servomotor driver control interface. The INP and ALM are used for feedback of the servo driver status, ERC is used to reset the servo driver's deviation counter under special conditions, VON is a general purpose output signal, and RDY is a general purpose input signal. General purpose means that the processing of the signal is not a hardware built-in procedure. The hardware processes INP, ALM, and ERC signals according to pre-defined rules. For example, when receiving ALM signal, the card stops or decelerate to stop output pulses automatically. However, SVON and RDY act like common I/Os.

4.2.1 INP

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

Usually, servomotor drivers with a pulse train input has a deviation (position error) counter to detect the deviations between the input pulse command and feedback counter. The driver controls the motion of the servomotor to minimize the deviation until it becomes 0. Theoretically, the servomotor operates with some time delay from the command pulses. Likewise, when the pulse generator stops outputting pulses, the servomotor does not stop immediately but keeps running until the deviation counter is zero. Only after stopping does the servo driver send out the in-position signal (INP) to the pulse generator to indicate the motor has stopped running.

If you do not enable INP checking by **`_8164_set_inp()`** function, the motion completion from **`_8164_motion_done()`** function and INT signal are raised after all pulses are sent.

If you enable INP checking by **`_8164_set_inp()`** function, the motion completion from **`_8164_motion_done()`** function and INT signal will not be raised until the INP signal from servo driver is raised.

The in-position function can be enabled or disabled, and the input logic polarity is also programmable by the "inp_logic" parameter of

_8164_set_inp(). The INP signal status can be monitored by software with the function: **_8164_get_io_status()**.

Related 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 the ALM signal is a hardware built-in procedure, and it is designed to interact with the alarm signal of the servomotor driver.

The ALM signal is an output signal from servomotor driver. Usually, it is designated to indicate when something is wrong with the driver or motor.

The ALM pin receives the alarm signal output from the servo driver. The signal immediately stops the card from generating any further pulses or stops it after deceleration. If the ALM signal is in the ON status at the start of an operation, the card generates the INT signal and does not generate any command pulses. The ALM signal may be a pulse signal with a minimum time width of 5 microseconds.

Setting the parameters “alm_logic” and “alm_mode” of the **`_8164_set_alm`** function can alter the input logic of the ALM. Whether or not the 8164 is generating pulses, the ALM signal allows the generation of the INT signal. The ALM status can be monitored by using the software function: **`_8164_get_io_status()`**. The ALM signal can generate an IRQ if the interrupt service is enabled. Refer to section 4.7.

Related 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 the 8164. The processing of the ERC signal is a hardware built-in procedure, and it is designed to interact with the deviation counter clear signal of the servomotor driver.

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

1. Home return is completed
2. The end-limit switch is active
3. An alarm signal stops the OUT and DIR signals
4. The software operator issues an emergency stop command

Since the servomotor operates with some delay from the pulse generated from the 8164, it continues to move until the deviation counter of the driver is zero, even if the 8164 has stopped outputting pulses because of the ±EL signal or the completion of home return. The ERC signal allows immediate stopping of the servomotor by resetting the deviation counter to zero. The ERC signal is outputted as a one-shot signal. The pulse width is of time length defined by the function call **`_8164_set_erc()`**. The ERC signal will automatically be generated when the ±EL and ALM signal are turned on and the servomotor is stopped immediately.

Related functions:

- _8164_set_erc(): Refer to section 6.12

4.2.4 SVON and RDY

All 864 axes are equipped with SVON and RDY signals, which are general purpose output and input channels, respectively. The SVON is used to interact with the servomotor drivers as a Servo ON command, and RDY to receive the Servo Ready signal. There are no built-in procedures for SVON and RDY.

The SVON signal is controlled by the software function **`_8164_Set_Servo()`**.

RDY pins are dedicated for digital input usage. The status of this signal can be monitored using the software function **`_8164_get_io_status()`**.

Related 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 signal operations are described.

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

In any operation mode, if an ±EL signal is active during any moving condition, it causes the 8164 to stop automatically outputting pulses. If an SD signal is active during moving conditions, it causes the 8164 to decelerate. If operating in a multi-axis mode, it automatically applies to all related axes.

4.3.1 SD/PCS

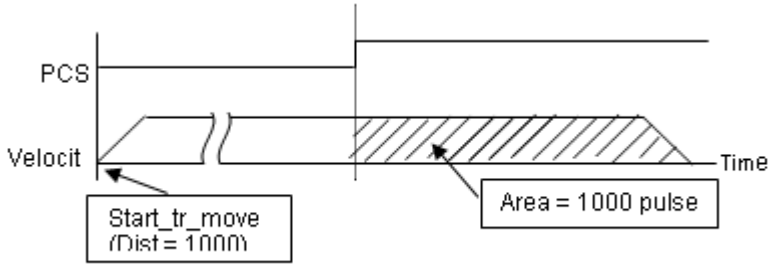
SD/PCS signal pins are available for each axis and acts as the input channel. It can be connected to a SD (Slow Down) or Position Change Signal (PCS). To configure the input signal type use the function **`_8164_set_sd_pin()`**.

When the SD/PCS pin is directed to a SD (the default setting), the PCS signal is kept at a low level and vice versa. Care must be taken with the logic attributes of the signal not being used.

The slow-down signals are used to force the output pulse (OUT and DIR) to decelerate to and then maintain the StrVel when it is active. The StrVel is usually smaller than MaxVel. This signal is useful in protecting a mechanism moving under high speeds toward the mechanism's limit. SD signal is effective for both plus and minus directions.

The ramping-down function can be enabled or disabled using the software function **`_8164_set_sd()`**. The input logic polarity, level operation mode, or latched input mode can also be set by this function. The signal status can be monitored using **`_8164_get_io_status()`**.

The PCS signal defines the starting point of current tr and sr motions. Refer to the chart below. The logic of PCS is configurable using **`_8164_set_pcs_logic()`**



Related 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 signal is used to stop the control output signals (OUT and DIR) when the end-limit is active. There are two possible stop modes: **stop immediately** and **decelerate to StrVel then stop**. To select either mode, use **`_8164_set_el()`**.

The PEL signal indicates the end-limit in the positive (plus) direction. MEL signal indicates the end-limit in negative (minus) direction. When the output pulse signals (OUT and DIR) is towards the positive direction, the pulse train will immediately stop when the PEL signal is asserted, where the MEL signal is meaningless, and vice versa. When the PEL is asserted, only a negative (minus) direction output pulse can be generated when moving the motor in a negative (minus) direction.

The EL signal can generate an IRQ if the interrupt service is enabled. Refer to section 4.7.

You can either use 'A' or 'B' type contact switches by setting the S1 dipswitch. The 8164 is delivered from the factory with all bits of S1 set to ON. The signal status can be monitored using the 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 operating in the home return mode. There are 13 home return modes (Refer to section 4.1.8), any one of 13 modes cam be selected using "home_mode" argument in the function **`_8164_set_home_config()`**. The logic polarity of the ORG signal level or latched input mode is also selectable using this function as well.

After setting the configuration for the home return mode with **`_8164_set_home_config()`**, the **`_8164_home_move()`** command can perform the home return function.

Related functions:

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

4.4 Counters

There are four counters for each card axis:

- **Command position counter**: counts the number of output pulses
- **Feedback position counter**: counts the number of input pulses
- **Position error counter**: counts the error between command and feedback pulse numbers.
- **General purpose counter**: The source can be configured as pulse output, feedback pulse, manual pulser, or CLK/2.
- **Target position recorder**, a software-maintained target position recorder, is discussed.

4.4.1 Command position counter

The command position counter is a 28-bit binary up/down counter. its input source is the output pulse from the 8164, thus, it provides accurate information of the current position. Note: the command position is different from target position. The command position increases or decreases according to the pulse output, while the target position changes only when a new motion command has been executed. The target position is recorded by the software, and needs manually resetting after a home move is completed.

The command position counter will clear (reset to “0”) automatically after a home move has completed. The function **`_8164_set_command()`** can be executed at any time to set a new command position value. To read current command position use **`_8164_get_command()`**.

Related functions:

- _8164_set_command(), _8164_get_command(): Refer to section 6.15

4.4.2 Feedback position counter

The card has a 28-bit binary up/down counter managing the present position feedback for each axis. The counter counts signal inputs from the EA and EB pins.

It accepts two kinds of pulse inputs: Plus and minus pulse inputs (CW/CCW mode), and; 90° phase shifted signals (AB phase mode). The 90° phase shifted signals maybe multiplied by a factor of 1, 2 or 4. 4x AB phase mode is the most commonly used in incremental encoder inputs. For example, if a rotary encoder has 2000 pulses per phase (A or B phase), then the value read from the counter is 8000 pulses per turn or –8000 pulses per turn depending on its rotating direction. These input modes can be selected using the **`_8164_set_pls_iptmode()`** function.

In cases where the application has not implemented an encoder, it is possible to set the feedback counter source to generate the output pulses, just as with the command counter. Thus, the feedback counter and the command counter will have the same value. To enable the counters to count the number of pulses inputted, set the "Src" parameter of the software function **`_8164_set_feedback_src()`** to **1**.

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

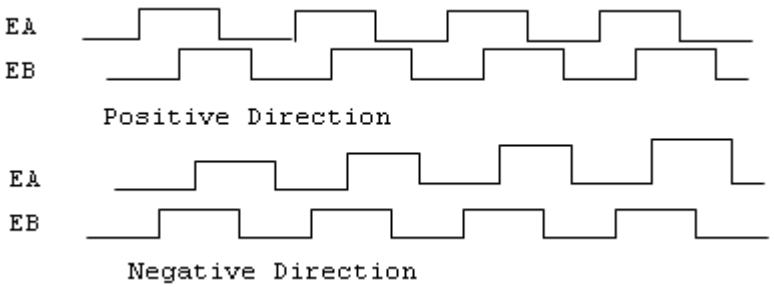
The pattern of pulses in this mode is the same as the ***Dual Pulse Output Mode*** in the Pulse Command Output section; except that the input pins are EA and EB.

In this mode, pulses from EA cause 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 a 90° phase leading or lagging in comparison with the EB signal. **Lead** or **lag** of phase difference between two signals is caused by the turning direction of the motor. The up/down counter counts up when the phase of EA signal leads the phase of EB signal.

The following diagram illustrates the waveform.



The index input (EZ) signals of the encoders are used as the ZERO reference. This signal is common on most rotational motors. EZ can be used to define the absolute position of the mechanism. The input logic polarity of the EZ signals is programmable using software function **`_8164_set_home_config()`**. The EZ signals status of the four axes can be monitored by **`get_io_status()`**.

The feedback position counter is automatically cleared to **0** after a home move is complete. Besides setting a position with the function call, **`_8164_set_position()`**, it can also be executed at any time to set a new position value. To read the current command position use **`_8164_get_position()`**.

Related functions:

- _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 the command position and the feedback position. It will add one count when the card outputs one pulse and subtracts one count when the card receives one pulse (from EA, EB). This is useful in detecting step-loses (stalls) in situations of a stepping motor when an encoder is applied.

Since the position error counter automatically calculates the difference between pulses outputted and pulses fed back, it is inevitable to get an error if the motion ratio is not equal to **1**.

To obtain a position error reading, use the function call **`_8164_get_error_counter()`**. To reset the position error counter, use the function call **`_8164_reset_error_counter()`**. The position error counter automatically clears to 0 after home move is complete.

Related function:

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

4.4.4 General purpose counter

The general purpose counter is very versatile and may be any of the following:

1. Pulse output – as a command position counter
2. Pulse input – as a feedback position counter
3. Manual Pulse input – Default status.
4. Clock – an accurate timer (9.8 MHz)

The default setting of the general purpose counter is set to manual pulse. (Refer to section 4.1.9 for a detailed explanation of manual pulsing). To change the source type, use the function **`_8164_set_general_counter()`**. To obtain the counter status, use the function **`_8164_get_general_counter()`**.

Related function:

- _8164_set_general_counter(),
  _8164_get_general_counter(): Refer to section 6.15

The table below summarizes all functions used for the different counter types.

Counter	Description	Counter Source	Function	Function description
Command	Counts the number of output pulses	Pulse output	_8164_set_command	Set a new value for command position
			_8164_get_command	Read current command position
Feedback	counts the number of input pulses	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	Counts the error between command and feedback pulse	EA/EB and Pulse output	_8164_get_error_counter	Gets the position error
			_8164_reset_error_counter	Resets the position error counter
General Purpose	General Purpose counter	Pulse output EA/EB manual pulse 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 provides target position information. For example, if the 8164 is operating in continuous motion with absolute mode, the target position lets the next absolute motion know the target position of previous one.

It is very important to understand how the software handles the target position recorder. Every time a new motion command is executed, the displacement is automatically added to the target position recorder. To ensure the correctness of the target position recorder, users need to manually maintain it in the following two situations using the function _8164_reset_target_pos():

1. After a home move completes
2. After a new feedback position is set

Relatied functions:

- _8164_reset_target_pos(): Refer to section 6.15

4.5 Multiple PCI-8164 Card Operation (PCI-8164 Only)

The software function library can support a maximum of 12 PCI-8164 cards. This means that up to 48 motors can be connected. Since the PCI-8164 is Plug-and-Play compatible, the base address and IRQ settings for card are are automatically assigned by the system BIOS when it is turned on. The base address and IRQ settings assigned by the BIOS can view by using the Motion Creator Tool.

When multiple cards are applied to a system, each card number must be noted. The card number of a PCI-8164 depends on the location on the PCI slot. They are numbered either from left to right, or right to left on the PCI slots. These card numbers will affect its corresponding axis number. Note that the axis number is the first argument for most functions called in the library. Hence, it is important to identify the slot number before writing any application programs. For example, if three PCI-8164 cards are plugged in to PCI slots, then the corresponding axis number on each card are:

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

Example: To accelerate Axis 3 of Card2 from 0 to 10000 pps in 0.5 sec for Constant Velocity Mode operation, the axis number is 6, and the code for the program is:

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

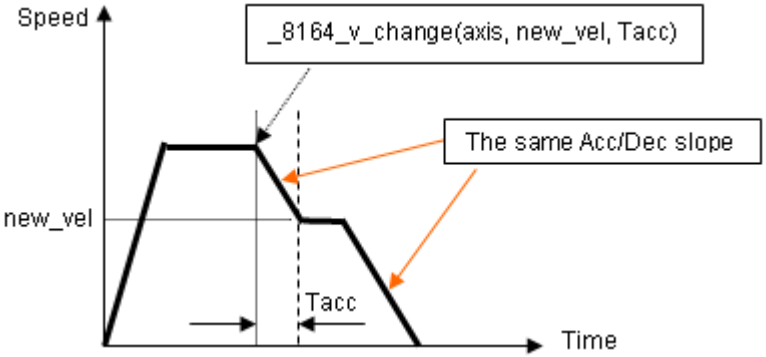
To determine the right card number, trial and error may be necessary before an application. The Motion Creator utility minimizes the search time.

For applications requiring many axes to move simultaneously on multiple PCI-8164 cards, connection diagrams in Section 3.12 should be followed to connect between CN4 connectors. Several functions listed in Section 6.8 may be useful when writing programs for such applications.

4.6 Change position or speed on the fly

The card provides the ability to change position or speed while an axis is moving. Changing speed/position on the fly means that the target speed/position can be altered after the motion has started. However, certain limitations do exist. Carefully study all constraints before implementing the on-the-fly function.

4.6.1 Change speed on the fly



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

The following functions are related to changing speed on the fly:

_8164_v_change() – change the MaxVel on the fly

_8164_cmp_v_change() –change velocity when the 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

The first four functions can be used for changing speed during a single axis motion. Functions **`_8164_sd_stop()`** and **`_8164_emg_stop()`** are used to decelerate the axis speed to **0**.

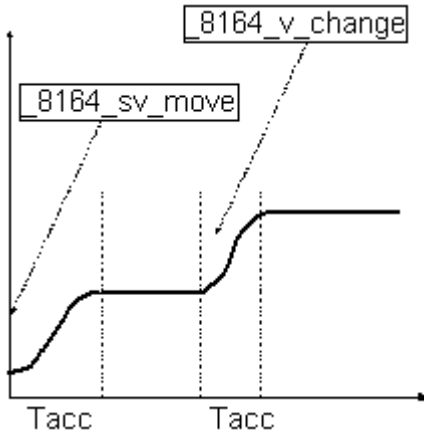
_8164_fix_speed_range() is necessary before any **_8164_v_change()** function, and **_8164_unfix_speed_range()** releases the speed range constrained by **_8164_fix_speed_range()**.

The function **_8164_cmp_v_change()** almost has the same function as **_8164_v_change()**, except **_8164_cmp_v_change()** acts only when a general comparator comes into existence. Refer to section 4.4.4 for more details about the general comparator.

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

Theory of **_8164_v_change()**:

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



Constraints of _8164_v_change():

In positioning mode, when changing to a higher velocity, there must be enough remaining pulses to decelereate after reaching new velocity, else the **`_8164_v_change()`** will return an error and the velocity remains unchanged.

For example:

A trapezoidal relative motion is applied:

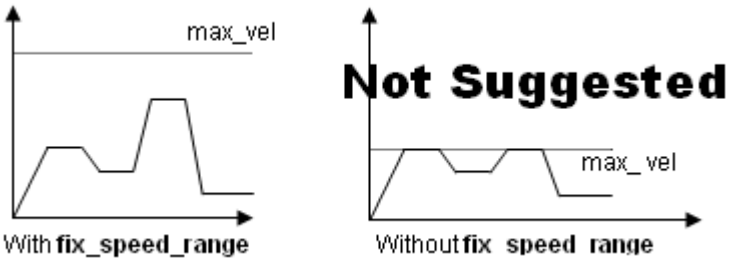
```
_8164_start_tr_move(0,10000,0,1000,0.1,0.1).
```

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

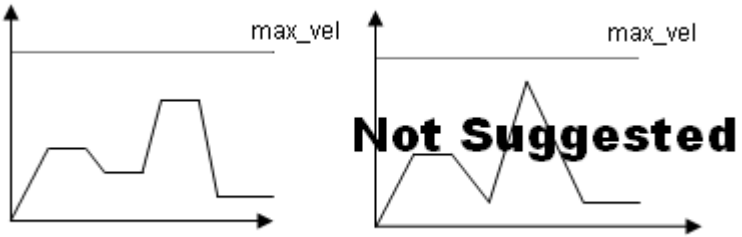
At 5000 pulses, _8164_v_change(0,NewVel,Tacc) is applied.

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. To set the maximum velocity, the function **`_8164_fix_speed_range()`** must be used in order for the function **`_8164_v_change()`** to work correctly. If **`_8164_fix_speed_range()`** is not applied, MaxVel set by **`_8164_v_move()`** or **`_8164_start_ta_move()`** automatically becomes the maximum velocity, where _8164_v_change() can not be exceeded.

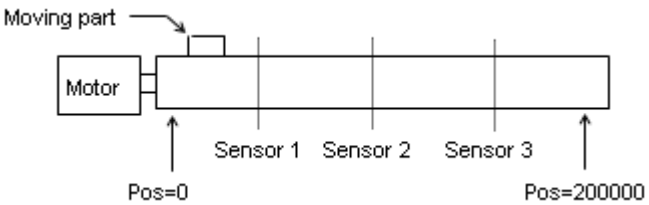


2. During the acceleration or deceleration period, using **`_8164_v_change()`** is not recommended. Even if it does work in most cases, the acceleration and deceleration time is not guaranteed.



Example:

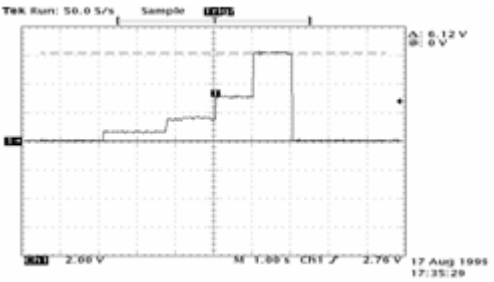
There are three speed change sensors during an absolute move for 200000 pulses. Initial maximum speed is 10000 pps. Change to 25000 pps if Sensor 1 is touched. Change to 50000 pps if Sensor 2 is touched. Change to 100000 pps if Sensor 3 is touched. Then the code for this application and the resulting 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 another 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);
}
```

The information of the three sensors is acquired from another I/O card and the resulting velocity profile from experiment is shown below:



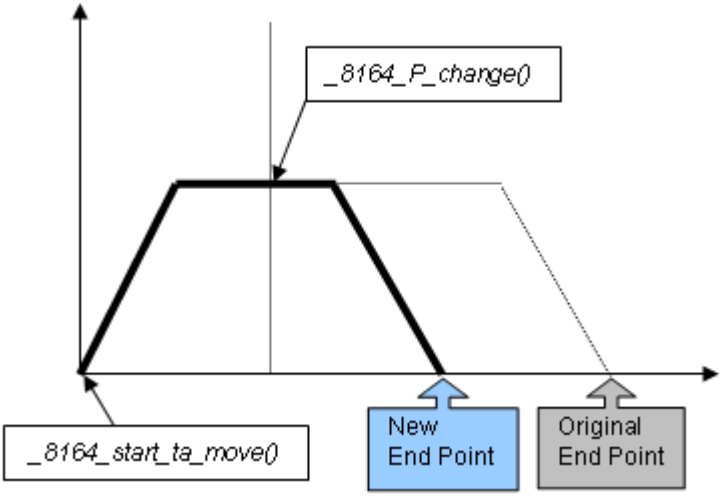
Related functions:

- _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

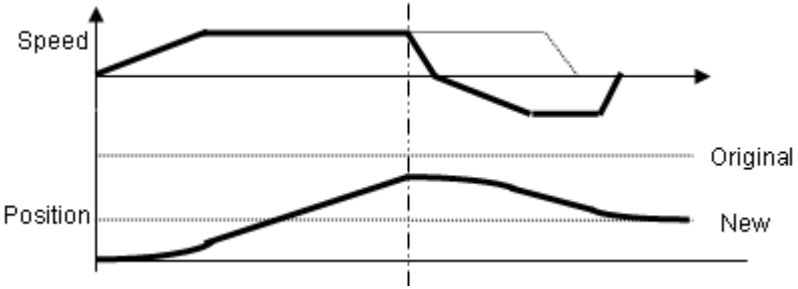
4.6.2 Change position on the fly

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



Theory of _8164_p_change():

The **`_8164_p_change()`** is only applicable for the **`_8164_start_ta_move()`**, and **`_8164_start_sa_move()`** functions. This function changes the target position, defined originally by these two functions. After changing position, the axis will move to the new target position and totally disregard the original position. If the new position is in the passed path, it will cause the axis to decelerate and eventually stop, then reverse, as shown in the chart. The acceleration and deceleration rate, and StrVel and MaxVel are kept the same as the original setting.



Constraints of _8164_p_change():

1. **`_8164_p_change()`** is only applicable on single-axis absolute pre-set motion, i.e. **`_8164_start_ta_move()`**, and **`_8164_start_sa_move()`** only.
2. Position change during the deceleration period is not allowed.
3. There must be enough distance between the new target position and current position where **`_8164_p_change()`** is executed because the 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 with a maximum velocity of 1000 PPS. The necessary number of pulses to decelerate is 0.5*1000*1 = 500.

At position “CurrentPos,” **`_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	

Related function:

- _8164_p_change(): refer to section 6.6

4.7 Position compare and Latch

The card provides position comparison functions on axes 0 and 1, and position latching functions on axes 2 and 3. The comparison function is used to output a trigger pulse when the counter reaches a preset value set by the user. CMP1 (axis 0) and CMP2 (axis 1) are used as a comparison trigger. The latch function is used to capture values on all 4 counters (refer to section 4.4) at the instant the latch signal is activated. LTC3 (axis 2) and LTC4 (axis 3) are used to receive latch pulses.

4.7.1 Comparators of the 8164

There are 5 comparators for each axis of the card. Each comparator has its unique functionality. Below is a table for comparison:

Comparator No	Compare Counter	Description	Function Related
Comparator 1	Command / Position	Soft Limit (+) **(Refer to section 4.9)**	*_8164_set_*softlimit *_8164_*enable_softlimit *_8164_*diable_softlimit
Comparator 2	Command / Position	Soft Limit (-) **(Refer to section 4.9)**	
Comparator 3	Error Counter	Step-losing detection	*_8164_error_counter_check*
Comparator 4	Any	General- purposed	*_8164_set_general_comparato*
Comparator 5 (Only Axis 0 & 1)	Any	Position compare function (Trigger)	*_8164_set_trigger_comparator* *_8164_build_compare_function* *_8164_build_compare_table* *_8164_set_auto_compare*

Note: Only comparator 5 has the ability to trigger an output pulse via the CMP pin.

Comparators 1 and 2 are used for soft limits. Refer to section 4.9. Comparator 3 is used to compare with the position error counter. It is useful for detecting if a stepping motor has lost any pulses. To enable/disable the step-losing detection, or set the allowable tolerance use **`_8164_set_error_counter_check()`**

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

Comparator 4 is a general purpose comparator, which will generate an 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 the function **`_8164_set_general_comparator()`**.

4.7.2 Position compare with trigger output

The 5th comparator, whose comparing source can be feedback or command position counter, performs the position compare function. Only the first 2 axes (0 and 1) can do a position comparison with trigger output. The position comparison function triggers a pulse output via the CMP pin, when the comparing condition comes into existence.

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

```
_8164_set_trigger_comparator (AxisNo, CmpSrc,
     Method, Data)
```

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

Continuously comparison with trigger output

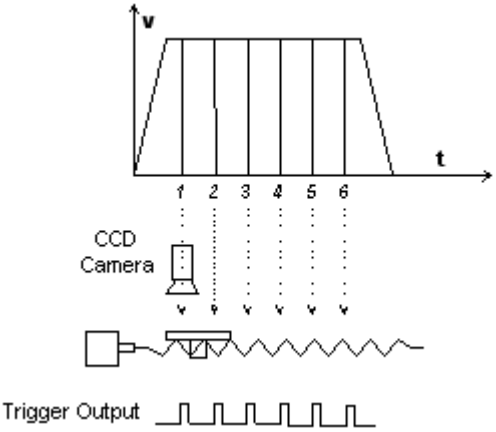
To compare multiple data continuously, functions for building comparison tables are provided and are shown below:

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

Note: Turn off all interrupt function and reduce accessing 8164 like get_position or issuing other commands when continuously comparison functions are running.

The first function builds a comparison list using start and end points and constant intervals. The second function builds on an arbitrary comparison table (data array). The third function is a source comparing selection function. Set this parameter to 1 to use the FIFO mode. Once it is set, the compare mechanism will start. Users can check current values used for comparison using the function_8164_check_compare_data():

Example: Using the continuous position comparison function.



In this application, the table is controlled by the motion command, and the CCD camera is controlled by the position comparison output of the 8164. An image of the moving object is easily obtained.

Working Spec: 34000 triggering points per stroke, trigger speed is 6000 pts/sec)

Program settings:

- Table starts moving from 0 to 36000
- Compare points are on 1001 35000, total 34000 pts, points to points interval = 1 pulse
- Moving Speed is 6000 pps
- Compare condition is “=“

Program code:

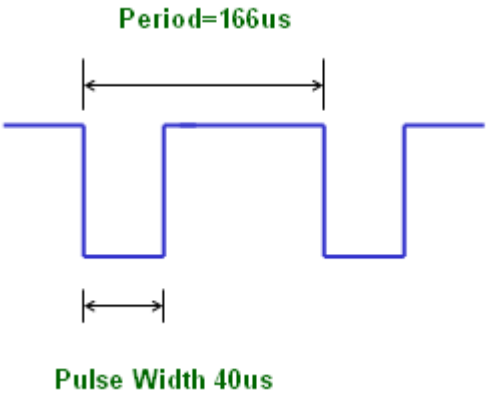
```
_8164_set_trigger_comparator(0, 1, 1, 1001);
_8164_build_compare_function(0, 1001, 35000, 1,
     1);
_8164_set_auto_compare(0, 1);
_8164_start_tr_move(0, 36000, 0, 6000, 0.01,
     0.01);
```

Monitoring or Check the current compare data:

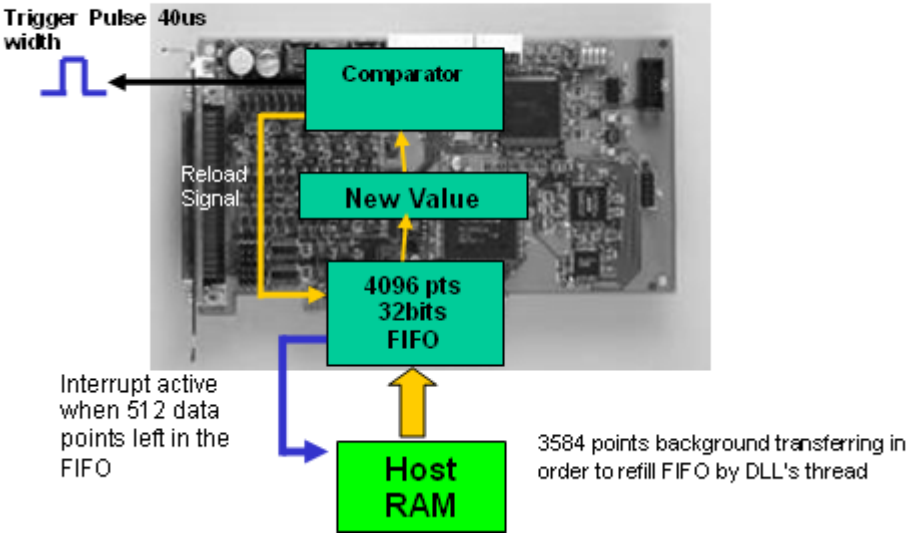
```
_8164_check_compare_data(0, 5, *CurrentData);
```

Users can use this function to check if auto-trigger is running.

Results:



The compare mechanism is shown below:



The **Value** block in this figure is the position where the comparison occurs, and where the data can be checked by using **`_8164_check_compare_data()`**.

Note that at the final compared point, load an After-final point into the Value block. Fill a dummy point into the comparison table array at the final position. This value must be far enough from the table's stroke.

If using **`_build_compare_function()`**, a dummy "after-final" point is automatically loaded. This value is equal to (End point + Interval x Total counts) x moving ratio.

Related functions:

- _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

The position latch is different than the position compare function in the following way: the position compare function triggers a pulse output via the CMP, when the comparing condition comes into existence, the position latch function receives pulse inputted via the LTC, and then captures all data in all counters at that instant (refer to section 4.4). The latency between the occurring latch signal and the finish position of the captured data is extremely short as the latching procedure is done by hardware. Only axes 2 and 3 can perform a position latch function. LTC3 (axis 2) and LTC4 (axis 3) are used to receive latch pulses.

To set the latch logic use **`_8164_set_ltc_logic()`**.

To obtain the latch values of the counters use **`_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.

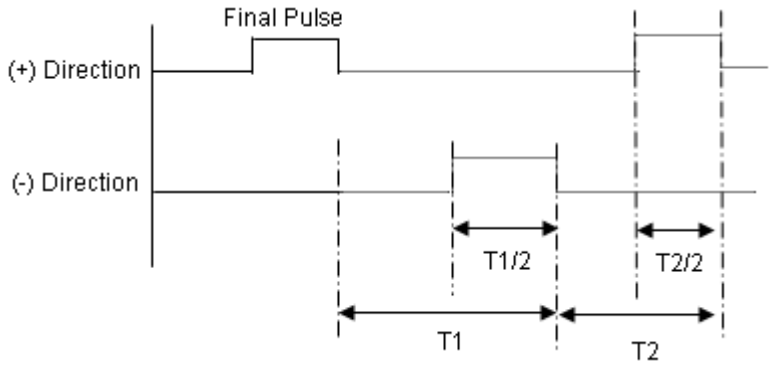
Related function:

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

4.8 Hardware backlash compensator and vibration suppression

Whenever direction change has occurred, the card outputs a backlash corrective pulse before sending the next command. The function **`_8164_backlash_comp()`** sets the pulse number.

In order to minimize vibration when a motor stops, the card can output a single pulse for a negative direction and then single pulse for a positive direction right after completion of a command movement. Refer to the timing chart below, the **`_8164_suppress_vibration()`** function is used to set T1 and T2.



Related function:

- _8164_backlash_comp(), _8164_suppress_vibration()

Refer to section 6.6

4.9 Software Limit Function

The card provides two software limits for each axis. The soft limit is extremely useful in protecting a mechanical system as it works like a physical limit switch when correctly set.

The soft limits are built on comparators 1 and 2 (Refer to section 4.7.1) and the comparing source is the command position counter.

A preset limit value is set in comparators 1 and 2, then, when the command position counter reaches the set limit value, the card reacts by generating the stop 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 only applied to the command position and not the feedback position (Refer to 4.4). In cases where the moving ratio is not equal to **1**, it is necessary to manually calculate its corresponding command position where the soft limit would be, when using **`_8164_set_softlimit()`**.

Related functions:

- _8164_set_softlimit(), _8164_enable_softlimit(), _8164_diable_softlimit()

Refer to section 6.16

4.10 Interrupt Control

The 8164 motion controller can generate an INT signal to the host PC. The parameter, “intFlag,” of the software function **`_8164_int_control()`**, can enable/disable the interrupt service.

After a interrupt occurs, the function **`_8164_get_int_status()`** is used to receive the INT status, which contains information about the INT signal. The INT status of the 8164 comprises of two independent parts: **`error_int_status`** and **`event_int_status`**. The **`event_int_status`** recodes the motion and comparator event under normal operation. This INT status can be masked by **`_8164_set_int_factor()`**. The error_int_status is for abnormal stoppage of the 8164 (i.e. EL, ALM, etc.). This INT cannot be masked. The following are the definitions of the two **`int_status`**:

error_int_status: can be masked by function call **`_8164_int_factor()`**

Bit	Description
0	+Soft Limit on and stop
1	-Soft Limit on and stop
2	(Reserved)
3	General Comparator on and stop
4	(Reserved)
5	+End Limit on and stop
6	-End Limit on and stop
7	ALM happen and stop
8	CSTP, Sync. stop on and stop
9	CEMG, Emergency on and stop
10	SD on and slow down to stop
11	(Reserved)
12	Interpolation Error and stop
13	Other axis stop on Interpolation
14	Pulse input buffer overflow and stop
15	Interpolation counter overflow
16	Encoder input signal error
17	Pulse input signal error
11-31	(Reserved)

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

Bit	Description
0	Normal Stop
1	Next command starts
2	Command pre-register 2 is empty
3	(Reserved)
4	Acceleration Start
5	Acceleration End
6	Deceleration Start
7	Deceleration End
8	Position Soft Limit On
9	Negative Soft Limit On
10	Error Comparator Compared
11	General Comparator compared
12	Trigger Comparator compared
13	(Reserved)
14	Counter Latched for axis2,3
15	ORG Input and Latched
16	SD on
17	(Reserved)
18	(Reserved)
19	CSTA, Sync. Start on
20-30	(Reserved)
31	Axis Stop Interrupt controlled by continuous motion command

Use Events to handle interrupts under Windows

To detect an interrupt signal from the card in Windows, you must first create an events array, then use the functions provided by the card to obtain the interrupt status. A sample program is listed below:

Steps:

1. Define a Global Value to deal with interrupt events. Each event is linked to an axis

```
HANDLE hEvent[4];
```

2. Enable interrupt event service and setup interrupt factors and enable interrupt channel

```
_8164_int_enable(0,hEvent);
_8164_set_int_factor(0,0x01); // Normal Stop
     interrupt
_8164_int_control(0,1);
```

3. Start move command

```
_8164_start_tr_move(0,12000,0,10000,0.1,0.1);
```

4. Wait for axis 0 interrupt event

```
STS=WaitForSingleObject(hEvent[0],15000);
ResetEvent(hEvent[0]);

if( STS==WAIT_OBJECT_0 )
{
_8164_get_int_status(0, &error, &event);
if( event == 0x01 ) …… ; // Success
}
else if( STS==WAIT_TIME_OUT)
{
     // Time out, fail
}
```

8164 Interrupt Service Routine (ISR) with DOS

A DOS function library is included with the card for developing applications under DOS environment. This library also includes a few functions to work with the ISR. It is highly recommended that programs be written according to the following example for applications working with the ISR. Since the PCI bus has the ability to do IRQ sharing when multiple cards are installed, each card must have a corresponding ISR. The library provided have the names of the ISR fixed, for example: _8164_isr0(void), _8164_isr1(void)…etc. A sample program is described below. It assumes that two cards are present in the system, axes 1 and 5 are requested to work with the 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 on 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){}
```

Related functions:

- _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

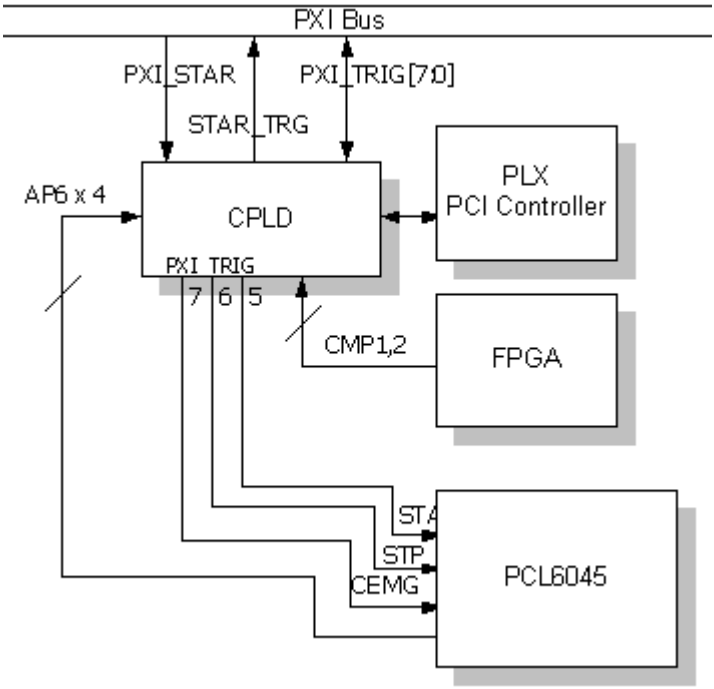
Refer to section 6.14

4.11 PXI Trigger Bus (PXI-8164 only)

There are eight general trigger channels in PXI bus. They are called PXI_TRG[0] to PXI_TRG[7]. All channels can be programmed as input or output. These trigger channels are for inter-board control synchronization.

For example, if there are two PXI cards, card A and card B in one system, Card A will send a trigger signal through PXI_TRG to card B and card B is waiting for this signal to do some action. There is another dedicated channel for trigger input and output, They are called PXI_STAR(input) and STAR_TRG(output). They can not be programmed as input or ouput individually. They can only be enabled or disabled.

This section describes the PXI_TRG's arrangement in PXI-8164 and how it is used. The following figure is the block diagram of PXI bus.



If you want to connect STA, STP, and CEMG to the PXI trigger bus, they can only set it to channel 5, 6, and 7, individually. STA may also be connected to PXI_START by function selection. Also, PXI_START's status could be read back through this function.

If you want to connect CMP1, CMP2 and AP6 to PXI trigger bus, they can set it to one of the PXI_TRG[7:0] by setting them as output. There is no dedicated channel for these two signals. Any sources active will force all of the PXI_TRG output. You may also enable STAR_TRG triggering.

If you want to control the PXI trigger bus' output by software, use the software trigger mode for these channels. All of the PXI_TRIG[7:0] could be used for this mode as output. Also, it can be set as input and may be readback in any time.

Related section: 6.22

5 Motion Creator

After installing the hardware (Chapters 2 and 3), it is necessary to correctly configure all cards and double-check the system before running. This chapter gives guidelines for establishing a control system and manually testing the cards to verify correct operation. The Motion Creator software provides a simple yet powerful means to setup, configure, test, and debug a motion control system that uses the PCI-/MPC-/PXI-8164 cards.

Note: Motion Creator is only available for Windows® 95/98 or Windows® NT/2000/XP operating system and a monitor with a screen resolution of 800x600 or higher. Motion Creator does not DOS environments.

5.1 Execute Motion Creator

After installing the software drivers for the card in Windows® 95/98/NT/2000/XP, the motion creator program is located at <chosen path>\PCI-Motion\MotionCreator. To launch the program, double-click on the executable file or click **Start** > **Program Files** > **PCI-Motion** > **MotionCreator**.

5.2 Notes on Motion Creator

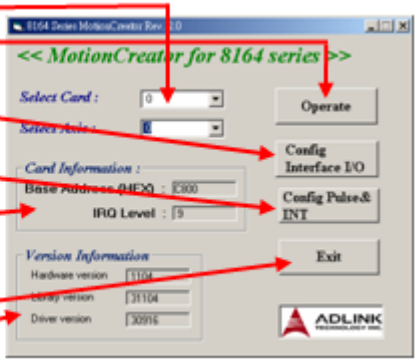
1. Motion Creator is a program written in VB 5.0, and is available only for Windows® 95/98/NT/2000/XP OS and a monitor with a screen resolution of 800x600 or higher. Motion Creator does not support DOS.
2. Motion Creator enables you to save settings and configurations for the card(s). Saved configurations are automatically loaded the next time Motion Creator is launched. The ***8164.ini*** and ***8164MC.ini*** files in **Windows root directory** saves all settings and configurations.
3. To duplicate configurations from one system to another, copy 8164.ini and 8164MC.ini into the Windows root directory.
4. When multiple cards use the same saved Motion Creator configuration files, the DLL function call **`_8164_config_from_file()`** can be invoked within a user developed program. This function is also available in DOS environment.

5.3 Using Motion Creator

5.3.1 Main Menu

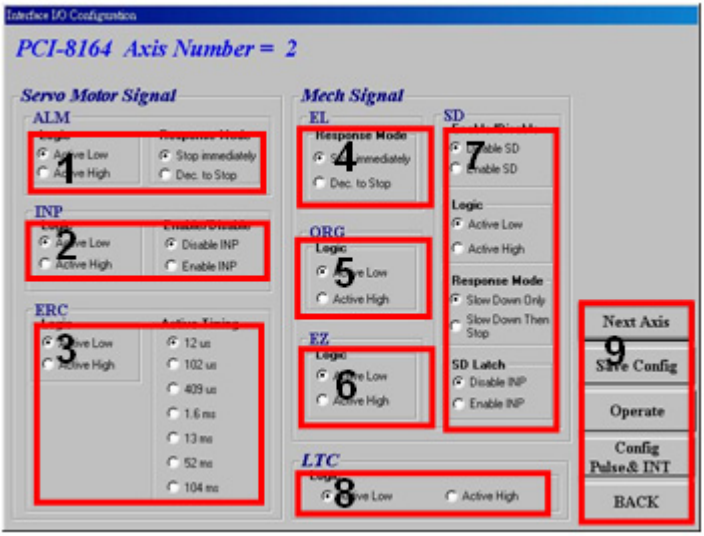
The main menu appears after launching the Motion Creator. Refer to the illustration on the next page for the description of the main menu functions.

- Select operating card and axis
- Go to **operation** menus (refer to section 5.3.4)
- Go to **Interface I/O** configuration menus (refer to section 5.3.2)
- Go to **Pulse & INT** configuration menus (refer to section 5.3.3)
- Show card information. Related function are : *_8164_get_base_addr(), _8164_get_irq_channel()*.
- Exit Motion Creator
- Version Inforamtion



5.3.2 Interface I/O Configuration Menu

This menu configures EL, ORG, EZ, ERC, ALM, INP, SD, and LTC.

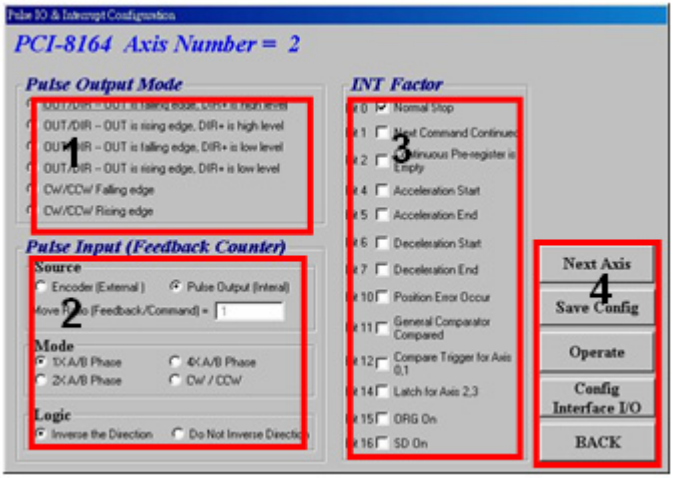


1. **ALM Logic and Response mode**: Selects the logic and response modes of ALM signal. The related function call is **`_8164_set_alm()`**.
2. **INP Logic and Enable/Disable selection**: Selects the logic, and enables/disables the INP signal. The related function call is **`_8164_set_inp()`**
3. **ERC Logic and Active timing**: Selects the Logic and Active timing of the ERC signal. The related function call is **`_8164_set_erc()`**.
4. **EL Response mode**: Selects the response mode of the EL signal. The related function call is **`_8164_set_el()`**.
5. **ORG Logic**: Selects the logic of the ORG signal. The related function call is **`_8164_set_home_config()`**.
6. **EZ Logic**: Selects the logic of the EZ signal. The related function call is **`_8164_set_home_config()`**.
7. **SD Configuration**: Configures the SD signal. The related function call is **`_8164_set_sd()`**.

8. **LTC Logic**: Selects the logic of the LTC signal. The related function call is **`_8164_set_ltc_logic()`**.
9. **Buttons**:
   - ▷ **Next Axis**: Changes the operating axis.
   - ▷ **Save Config**: Saves current configuration to 8164.ini.
   - ▷ **Operate**: Go to the operation menu. Refer to section 5.3.4.
   - ▷ **Config Pulse & IN**T: Go to the Pulse I/O and Interrupt Configuration menu. Refer to section 5.3.3.
   - ▷ **Back**: Returns to the main menu.

5.3.3 Pulse I/O and Interrupt Configuration Menu

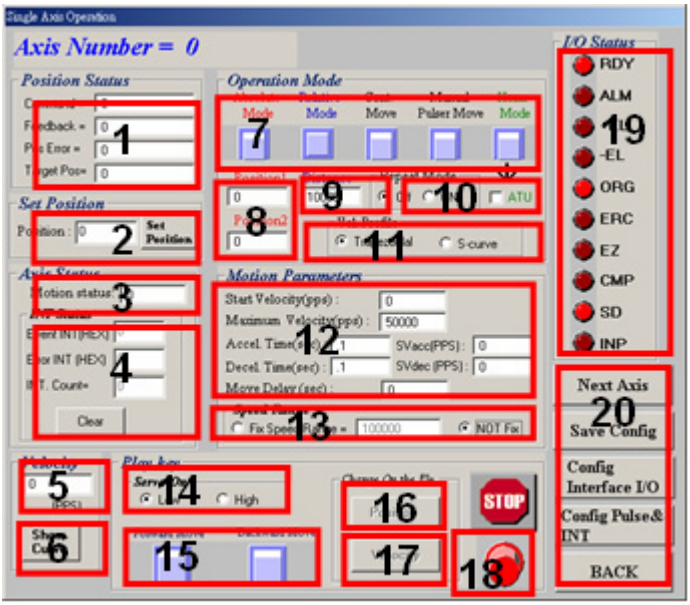
This menu configures the pulse input/output and move ratio and INT factor.



1. **Pulse Output Mode**: Selects the output mode of the pulse signal (OUT/ DIR). The related function call is **`_8164_set_pls_outmode()`**.
2. **Pulse Input**: Sets the configurations of the Pulse input signal(EA/EB). The related function calls are **`_8164_set_pls_iptmode()`**, **`_8164_set_feedback_src()`**.
3. **INT Factor**: Selects factors to initiate the event int. The related function call is **`_8164_set_int_factor()`**.
4. **Buttons**:
   - ▷ **Next Axis**: Changes the operating axis.
   - ▷ **Save Config**: Saves the current configuration to 8164.ini.
   - ▷ **Operate**: Go to the operation menu. Refer to section 5.3.
   - ▷ **Config Pulse & INT**: Go to the Pulse I/O and Interrupt Configuration menu. Refer to section 5.3.
   - ▷ **Back**: Return to the main menu.

5.3.4 Operation menu:

This menu changes the settings for a selected axis, including velocity mode motion, preset relative/absolute motion, manual pulse move, and home return.

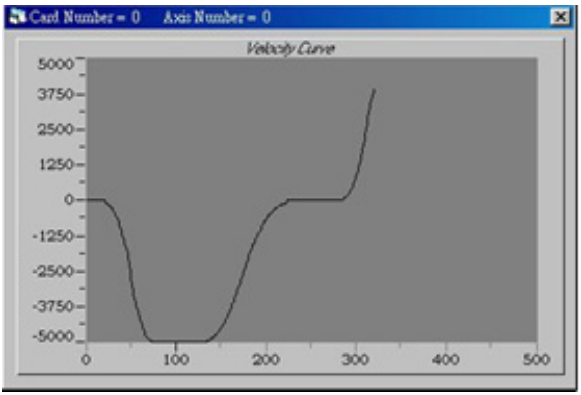


1. **Position**:
   - ▷ Command: displays the value of the command counter. The related function is **`_8164_get_command()`**.
   - ▷ Feedback: displays the value of the feedback position counter. The related function is **`_8164_get_position()`**
   - ▷ Pos Error: displays the value of the position error counter. The related function is **`_8164_get_error_counter()`**.
   - ▷ Target Pos: displays the value of the target position recorder. The related function is **`_8164_get_target_pos()`**.
2. **Position Reset**: Sets all positioning counters to a speci-

fied value. The related functions are:

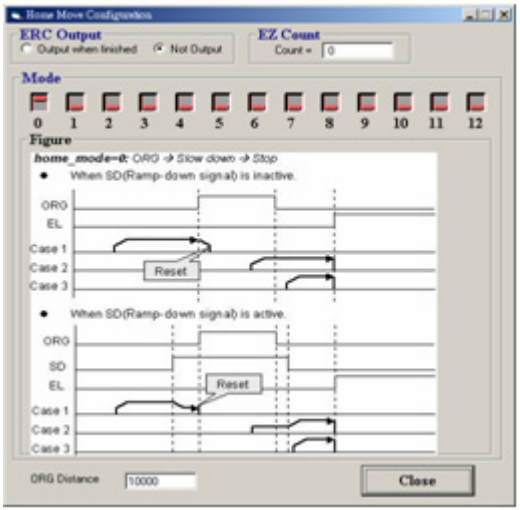
- `_8164_set_position()`
- `_8164_set_command()`
- `_8164_reset_error_counter()`
- `_8164_reset_target_pos()`

3. **Motion Status**: Displays the returned value of the `_8164_motion_done` function. The related function is `_8164_motion_done()`.
4. **INT Status**:
   - **Event**: displays the event_int_status (in hexadecimal). The related function is `_8164_get_int_status()`.
   - **Error**: displays the error_int_status (in hexadecimal). The related function is `_8164_get_int_status()`.
   - **Count**: total count of interrupt.
   - **Clear Button**: clears all INT status and counter to **0**.
5. **Velocity**: The absolute value of velocity in units of PPS. The related function is `_8164_get_current_speed()`.
6. **Show Velocity Curve Button**: Opens up a window showing a velocity vs. time curve. In this curve, every 100 ms, a new velocity data point will be added. Click the same button to close the window. To clear data, click on the curve.



7. **Operation Mode**: Select operation mode.
   - ▷ **Absolute Mode**: Sets the Position1 and Position2 as absolution target positions for motion. The related functions are **`_8164_start_ta_move()`**, _8164_start_sa_move().
   - ▷ **Relative Mode**: Uses Distance 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 Pulse motion. Clicking this button invokes the manual pulse configuration window.
   - ▷ **Home Mode**: Home return motion. Clicking this button invokes the home move configuration window. The related function is **`_8164_set_home_config()`**. *If the ATU check box is checked, it will execute auto homing when motion starts.*
   - ▷ **ERC Output**: Select to send the ERC signal when home move is completed.
   - ▷ **EZ Count**: Sets the EZ count number, which is effective on certain home return modes.
   - ▷ **Mode**: Select from 13 home return modes.
   - ▷ **Home Mode figure**: The figure below explains the actions of the individual home modes.
   - ▷ **Close**: Click this button to close the window.

▷ **ORG Distance**: The length when ORG is ON



8. **Position**: Sets the absolute position for Absolute Mode. This is only available when the Absolute Mode is selected.
9. **Distance**: Sets the relative distance for Relative Mode. This is only available when Relative Mode is selected.
10. **Repeat Mode**: When you select On, the motion will become a repeat mode (forward<->backward or position1<->position2). This is only available when Relative Mode or Absolute Mode is selected.
11. **Vel. Profile**: Selects the velocity profile. Both Trapezoidal and S-Curve are available for Absolute Mode, Relative Mode, and Cont. Move.
12. **Motion Parameters**: Sets the parameters for single axis motion. This parameter is meaningless if Manual Pulser Move is selected, since the velocity and moving distance is decided by pulse input.
    ▷ **Start Velocity**: Sets the start velocity of motion in units of PPS. In Absolute Mode or Relative Mode, only the value is effective. For example, -100.0 is the same as

100.0. In Cont. Move, both the value and sign are effective.
–100.0 means 100.0 in the minus direction.

- ▷ **Maximum Velocity**: Sets the maximum velocity of motion in units of PPS. In Absolute Mode or Relative Mode, only the value is effective. For example, -5000.0 is the same as 5000.0. In Cont. Move, both the value and sign is effective. –5000.0 means 5000.0 in the minus direction.
- ▷ **Accel. Time**: Sets the acceleration time in units of second.
- ▷ **Decel. Time**: Sets the deceleration time in units of second.
- ▷ **SVacc**: Sets the S-curve range during acceleration in units of PPS.
- ▷ **SVdec**: Sets the S-curve range during deceleration in unit sof PPS.
- ▷ **Move Delay**: This setting is available only when repeat mode is On. It causes the card to delay for a specified time before it continues to the next motion.

13. **Speed Range**: Sets the max speed of motion. When you select Not Fix, the Maximum Speed automatically becomes the maximum speed range, which can not be exceeded by on-the-fly velocity change.
14. **Servo On**: Sets the SVON signal output status. The related function is **`_8164_set_servo()`**.
15. **Play Key**:
    - ▷ **Left play button**: Clicking this button lets the card start to outlet pulses according to previous setting.
    - ▷ **In Absolute Mode**, this causes the axis to move to position1.
    - ▷ **In Relative Mode**, this causes the axis to move forward. In Cont. Move, this causes the axis to start to move according to the velocity setting.

- ▷ **In Manual Pulser Move**, this causes the axis to go into pulse move. The speed limit is the value set by Maximum Velocity.
- ▷ **Right play button**: Click this button to let the card start to outlet pulses according to previous setting.
- ▷ **In Absolute Mode**, this causes the axis to move to position.
- ▷ **In Relative Mode**, this causes the axis to move backwards.
- ▷ **In Cont. Move**, this causes the axis to start to move according to the velocity setting, but in the opposite direction.
- ▷ **In Manual Pulser Move**, this causes the axis to go into pulse move. The speed limit is the value set by Maximum Velocity.

16. **Change Position On The Fly Button**: When enabled, you can change the target position of the current motion. The new position must be defined in Position2. The related function is **`_8164_p_change()`**.
17. **Change Velocity On The Fly Button**: When enabled, you can change the velocity of the current motion. The new velocity must be defined in Maximum Velocity. The related function is **`_8164_v_change()`**
18. **Stop Button**: Clicking this button causes the card to decelerate and stop. The deceleration time is defined in Decel. Time. The related function is **`_8164_sd_stop()`**.
19. **I/O Status**: The status of motion I/O. Light-On means the motion I/O is active while Light-Off indicates inactivity. The related function is **`_8164_get_io_status()`**.

20. **Buttons**:

- ▷ **Next Axis**: Changes the operating axis.
- ▷ **Save Config**: Saves the current configuration to 8164.ini.
- ▷ **Config Pulse & INT**: Go to the Pulse IO and Interrupt Configuration menu. Refer to section 5.3.3
- ▷ **Config Interface I/O**: Go to the Interface I/O Configuration menu. Refer to section 5.3.2
- ▷ **Back**: Return to the main menu.

6 Function Library

This chapter describes the supporting software for the PCI-/MPC-/PXI-8164 card. You can use these functions to develop programs in C, C++, or Visual Basic. If Delphi is used as the programming environment, you need to manually transform the header files 8164.h.

6.1 List of Functions

Initialization Section 6.3

Function Name	Description
_8164_initial	Card initialization
_8164_initialx	Card initialization with I/O base address and IRQ Channel
_8164_close	Card Close
_8164_get_base_addr	Get base address of 8164
_8164_get_irq_channel	Get the 8164 card's IRQ number
_8164_delay_time	Delay execution of program for specified time in units of ms.
_8164_config_from_file	Configure 8164 cards according to configuration file i.e. 8164.ini, which is created by Motion Creator.
_8164_version_info	Check the hardware and software version

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
_8164_verify_speed	Check the min/max acceleration time under max 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 the SD/PCS pin
_8164_backlash_comp	Set backlash corrective pulse for compensation
_8164_suppress_vibration	Set vibration suppressing timing
_8164_set_idle_pulse	Set suppress vibration idle pulse counts
_8164_set_fa_speed	Set FA speed for home mode
_8164_dwell_move	Start a dwell move

Linear Interpolated Motion Section 6.7

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 an 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 an 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 an 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 an 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 an absolute 2-axis linear interpolation for any 2 axes, with trapezoidal profile
_8164_start_sa_line2	Begin an 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 an absolute 3-axis linear interpolation with trapezoidal profile
_8164_start_sa_line3	Begin an absolute 3-axis linear interpolation with S-curve profile,
_8164_start_tr_line4	Begin a relative 4-axis linear interpolation with trapezoidal profile

Function Name	Description
_8164_start_sr_line4	Begin a relative 4-axis linear interpolation with S-curve profile
_8164_start_ta_line4	Begin an absolute 4-axis linear interpolation with trapezoidal profile
_8164_start_sa_line4	Begin an absolute 4-axis linear interpolation with S-curve profile
_8164_set_axis_option	Choose interpolation speed mode

Circular Interpolation Motion Section 6.8

Function Name	Description
_8164_start_a_arc_xy	Begin an 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 an absolute circular interpolation for Z & U
_8164_start_r_arc_zu	Begin a relative circular interpolation for Z & U
_8164_start_a_arc2	Begin an 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
_8164_start_tr_arc_xyu	Begin a t-curve relative arc with U axis sync.
_8164_start_ta_arc_xyu	Begin a t-curve absolute arc with U axis sync.
_8164_start_sr_arc_xyu	Begin a s-curve relative arc with U axis sync
_8164_start_sa_arc_xyu	Begin a s-curve absolute arc with U axis sync
_8164_start_tr_arc_xy	Begin a t-curve relative circular interpolation for X & Y
_8164_start_ta_arc_xy	Begin a t-curve absolute circular interpolation for X & Y
_8164_start_sr_arc_xy	Begin a s-curve relative circular interpolation for X & Y
_8164_start_sa_arc_xy	Begin a s-curve absolute circular interpolation for X & Y
_8164_start_tr_arc_zu	Begin a t-curve relative circular interpolation for Z & U
_8164_start_ta_arc_zu	Begin a t-curve absolute circular interpolation for Z & U
_8164_start_sr_arc_zu	Begin a s-curve relative circular interpolation for Z & U
_8164_start_sa_arc_zu	Begin a s-curve absolute circular interpolation for Z & U
_8164_start_tr_arc2	Begin a t-curve relative circular interpolation for any 2 of the 4 axes
_8164_start_ta_arc2	Begin a t-curve absolute circular interpolation for any 2 of the 4 axes
_8164_start_sr_arc2	Begin a s-curve relative circular interpolation for any 2 of the 4 axes
_8164_start_sa_arc2	Begin a s-curve absolute 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
_8164_escape_home	Escape Home Function
_8164_home_search	Auto-Search Home Switch

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
_8164_set_pulser_ratio	Set manual pulser ratio for actual output pulse rate
_8164_pulser_r_line2	pulser mode for 2-axis linear interpolation
_8164_pulser_r_arc2	pulser mode for 2-axis arc interpolation

Motion StatusSection 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 operating mode

Motion I/O Monitoring Section 6.13

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

Interrupt Control Section 6.14

Function Name	Description
_8164_int_control	Enable/Disable INT service
_8164_int_enable	Enable event (For Windows only)
_8164_int_disable	Disable event (For Windows only)
_8164_get_int_status	Get INT Status (For Windows only)
_8164_link_interrupt	Set link to interrupt call back function (For Windows 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)
_8164_set_axis_stop_int	Enable axis stop int
_8164_mask_axis_stop_int	Mask axis stop int

Position Control and Counters Section 6.15

Function Name	Description
_8164_get_position	Get the value of the feedback position counter
_8164_set_position	Set the feedback position counter
_8164_get_command	Get the value of the command position counter
_8164_set_command	Set the command position counter
_8164_get_error_counter	Get the value of the position error counter
_8164_reset_error_counter	Reset the position error counter
_8164_get_general_counter	Get the value of the general counter
_8164_set_general_counter	Set the general counter
_8164_get_target_pos	Get the value of the target position recorder
_8164_reset_target_pos	Reset target position recorder
_8164_get_rest_command	Get remaining pulses until the 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
_8164_force_cmp_output	Force to output trigger pulses
_8164_set_compare_mode	A general function for setting comparator mode
_8164_set_compare_data	A general function for setting comparator data
_8164_set_rotary_counter	Set counter as a rotary counter

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

Multiple Axes Simultaneous Operation Section 6.18

Function Name	Description
_8164_set_tr_move_all	Multi-axis simultaneous operation setup
_8164_set_ta_move_all	Multi-axis simultaneous operation setup
_8164_set_sr_move_all	Multi-axis simultaneous operation setup
_8164_set_sa_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
_8164_set_sync_option	Optional sync options
_8164_set_sync_stop_mode	Set the stop mode when CSTOP signal is ON
_8164_set_sync_signal_source	Select internal sync. signal's source of axis
_8164_set_sync_signal_mode	Select internal sync. signal's condition

General-purposed TTL Output Section 6.19 (PCI-8164 Only)

Function Name	Description
_8164_d_output	Digital Output
_8164_get_dio_status	Get DO status

General-purposed DIO Section 6.20 (MPC-8164 Only)

Function Name	Description
_8164_write_do	Digital Output
_8164_read_di	Digital Input

General-purposed DIO Section 6.20 (PXI-8164 Only)

Function Name	Description
_8164_write_axis_do	Digital Output
_8164_read_axis_di	Digital Input

Card ID Section 6.21 (PXI-8164 Only)

Function Name	Description
_8164_enable_card_id	Enable card ID's function
_8164_check_card	Check if this Card ID exist

PXI Trigger Bus Section 6.22 (PXI-8164 Only)

Function Name	Description
_8164_get_pxi_trigger_value	Digital Output
_8164_set_pxi_trigger_value	Digital Input
_8164_enable_pxi_input	Enable PXI input channel
_8164_select_pxi_output	Select PXI output channel

6.2 C/C++ Programming Library

This section discusses the functions in detail. The function prototypes and some common data types are declared in **PCI-8164.H** or **MPC-8164.H**. It is recommended that you use these data types in your application programs. The following table shows the data type names and their range.

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

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

In a 'C' programming environment:

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

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

6.3 Initialization

@ Name

_8164_initial – Card Initialization

_8164_initialx – Card Initialization with I/O base address and IRQ channel

_8164_close – Card Close

_8164_get_base_addr – Get the base address of 8164_

_8164_get_irq_channel – Get the 8164 card's IRQ number

_8164_delay_time – delay execution of program for specified time in units of ms.

_8164_config_from_file – Configure 8164 card according to configuration file i.e. 8164.ini.

_8164_version_info – Check hardware and software version information

@ Description

_8164_Initial:

Initializes the card without assigning the hardware resources. All 8164 cards must be initialized by this function before calling other functions. The card uses this function in all platforms because it is Plug and Play compatible. The MPC-8164 uses this function in Windows® 98/NT/2000/XP.

_8164_initialx:

Initializes the cards with an I/O base address and IRQ channel. The MPC-8164 uses this function under DOS, Windows® CE, and Linux.

_8164_close:

Closes the card and releases its resources. This function must be called at the end of an application.

_8164_get_irq_channel:

Gets the card's IRQ number.

_8164_get_base_addr:

Get the card's base address.

_8164_delay_time:

Delays execution of program for specified time in units of ms.

_8164_config_from_file:

Loads the configuration of the card based on the specified file. By using *Motion Creator*, users can test and configure the card correctly. After pressing the **save config** button, the configuration is saved as 8164.ini in the Windows directory. By specifying it in the parameter, the configuration is automatically loaded.

When this function is executed, all 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
```

_8164_version_info:

Reads back version information.

@ Syntax

C/C++ (DOS, Windows 95/NT/2K/XP)

```
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_delay_time(I16 AxisNo, U32 MiniSec);
I16 _8164_config_from_file(char *filename);
I16 _8164_version_info(I16 CardNo, U16
     *HardwareInfo, U16
 *SoftwareInfo, U16 *DriverInfo);
```

Note: In RTX environments, the type of information changes to U32

Visual Basic (Windows 95/NT/2K/XP)

```
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_delay_time (ByVal AxisNo As Integer, ByVal
     MiniSec As Long) As Integer
B_8164_config_from_file(ByVal filename As
     string)as integer
B_8164_version_info (ByVal CardNo As Integer,
     HardwareInfo As Integer, SoftwareInfo As
     Integer, DriverInfo As Integer) As Integer
```

@ Argument

`*existCards`: Number of installed 8164 cards

`cardNo`: Card index number

`AxNo`: Specifies which axis to use in measuring the delay time

`*irq_no`: IRQ number of an specified 8164 card

`*base_addr`: Base address of an specified 8164 card

`*Filename`: The specified filename recording the configuration of 8164. This file is created by the Motion Creator.

*HardwareInfo: Hardware version readback in decimal

Digit 3	Digit 2	Digit 1	Digit 0
0: PCL-6045 1: PCL-6045A	PXI-8164 CPLD version	0:PCI-8164 1: MPC-8164 2: PCI-8164IL	0: CPLD A1, A2 3: CPLD A3 4:CPLD A3

*SoftwareInfo: Software library version readback in decimal

	Digit 4	Digit 3	Digit 2	Digit 1	Digit 0
Win32	3: Year 2003 1: Year 2004 2: Year 2005 0: Year 2006	Month:1~12		Day:01~31	
WinCE		Month + 12			
DOS		Month + 24			
DOSExt		Month + 36			
Linux		Month + 48			

*DriverInfo: Device driver version readback in decimal

	Digit 4	Digit 3	Digit 2	Digit 1	Digit 0
Win32	3: Year 2003 1: Year 2004 2: Year 2005 0: Year 2006	Month: 1~12		Day:01~31	
WinCE		Month + 12			
DOS		Month + 24			
DOSExt		Month + 36			
Linux		Month + 48			

@ 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:

Configures the output modes of command pulses. There are six modes for command pulse output.

_8164_set_pls_iptmode:

Configures the input modes of external feedback pulses. There are four types for feedback pulse input. This functionis only available when the **Src** parameter in **_8164_set_feedback_src()** function is enabled.

_8164_set_feedback_src:

If external encoder feedback is available in the system, enable the **Src** parameter in this function. The internal 28-bit up/down counter counts according to the configuration of the **_8164_set_pls_iptmode()** function. Otherwise, the counter will count the command pulse output.

@ Syntax

C/C++ (DOS, Windows 95/NT/2K/XP)

```
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/2K/XP)

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

pls_outmode: Setting of command pulse output mode

Value	Meaning	Notes
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 low level
4	CW/CCW	Falling edge
5	CW/CCW	Rising edge
6	4XA/B	Falling edge
7	4XA/B	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, Not inverse direction
- **pls_logic**=1, Inverse direction

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

_8164_verify_speed – get speed profile's minimum and maximum acc/dec time

@ Description

_8164_tv_move:

Accelerates an axis to the specified constant velocity with a trapezoidal profile. The axis continues 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 the velocity parameter.

_8164_sv_move:

Accelerates an axis to the specified constant velocity with a S-curve profile. The axis continues 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:

Changes the moving velocity with a trapezoidal profile or S-curve profile. Before calling this function, 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 an original motion profile. When using in S-curve, set the motion to pure S-curve. Refer to the function limitations on section 4.6.1 before using.

_8164_sd_stop:

Decelerates an axis to stop with a trapezoidal or S-curve profile. This function is also useful when a *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:

Immediately stops an axis. This function is also useful when a *preset move* (both trapezoidal and S-curve motion), *manual move*, or *home return* function is performed.

_8164_fix_speed_range:

Defines the speed range. It should be called before starting motion that may contains velocity changing.

_8164_unfix_speed_range:

Releases the speed range constrains.

_8164_get_current_speed:

Reads the current pulse output rate of a specified axis. It is applicable in any time in any operating mode.

_8164_verify_speed:

Verifies a speed profile's minimum and maximum accelerating time.

@ Syntax

C/C++ (DOS, Windows 95/NT/2K/XP)

```
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);
F64 _8164_verify_speed(F64 StrVel,F64 MaxVel,F64
     *minAccT,F64 *maxAccT, F64 MaxSpeed);
```

Visual Basic (Windows 95/NT/2K/XP)

```
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
B_8164_verify_speed Lib "8164.DLL" Alias
     "_8164_verify_speed" (ByVal StrVel As
     Double, ByVal MaxVel As Double, minAccT As
     Double, maxAccT As Double, ByVal MaxSpeed As
     Double) As Double
```

@ Argument

AxisNo: Designated axis number to move or stop

StrVel: Starting velocity in units of pulse per second

MaxVel: Maximum velocity in units of pulse per second

Tacc: Specified acceleration time in units of second

SVacc: Specified velocity interval in which S-curve acceleration is performed.

Note: SVacc = 0, for pure S-Curve

NewVel: New velocity in units of pulse per second

Tdec: specified deceleration time in units 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

_8164_set_idle_pulse – Set suppress vibration idle pulse counts

_8164_dwell_move – Begin a dwell move

_8164_set_fa_speed – Set FA speed in home mode

@ Description

General: The moving direction is determined by the sign of the **Pos** or **Dist** parameter. If the moving distance is too short to reach the specified velocity, the controller automatically lowers the MaxVel, and the Tacc, Tdec, VSacc, and VSdec become shorter while dV/dt(acceleration / deceleration) and d(dV/dt)/dt (jerk) remain unchanged.

_8164_start_tr_move:

Accelerates the axis from a starting velocity, slew at constant velocity, and decelerates to stop at the relative distance with trapezoidal profile. The acceleration and deceleration time is specified independently. This function does not let the program wait for motion completion but immediately returns control to the program.

_8164_start_ta_move:

Accelerates the axis from a starting velocity, slew at constant velocity, and decelerates to stop at the specified absolute position with trapezoidal profile. The acceleration and deceleration time is specified independently. This command does not let the program wait for motion completion, but immediately returns control to the program.

_8164_start_sr_move:

This function accelerates the axis from a starting velocity, slew at constant velocity, and decelerates to stop at the relative distance with S-curve profile. The acceleration and deceleration time is specified independently. This command does not let the program wait for motion completion, but immediately returns control to the program.

_8164_start_sa_move:

This function accelerate the axis from a starting velocity, slew at constant velocity, and decelerates to stop at the specified absolute position with S-curve profile. The acceleration and deceleration time is specified independently. This command does not let the program wait for motion completion but immediately returns control to the program.

_8164_set_move_ratio:

Configures the 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 is equal to 2.

_8164_p_change:

Changes the target position on the fly. Refer to the function limitations on section 4.6.2 before using.

_8164_set_pcs_logic:

Sets the logic of Position Change Signal (pcs). The PCS shares the same pin with the SD signal. Only when the SD/PCS pin is set to PCS by _8164_set_sd_pin, that this _8164_set_pcs_logic function becomes available.

_8164_set_sd_pin:

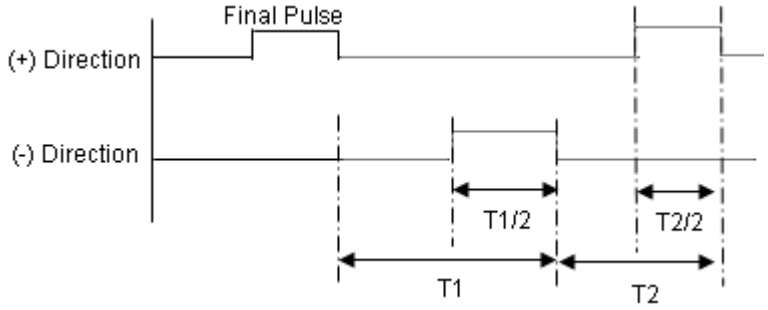
Sets the operating mode of the SD pin. The SD pin may be used as a Slow-Down signal input or as a Position Change Signal (PCS) input. Refer to section 4.3.1

_8164_backlash_comp:

Whenever direction change occurs, the 8164 outputs backlash corrective pulses before sending commands. This function sets the compensation pulse numbers. It uses FA speed when backlashing.

_8164_suppress_vibration:

Suppresses vibration of mechanical systems by outputting a single pulse for negative direction then single pulse for positive direction right after completion of command movement.



_8164_set_idle_pulse:

Delays acceleration from starting velocity. This outputs the counts of setting pulses at starting velocity then acceleration.

_8164_dwell_move:

Starts a dwell move that means the move does not cause real motion for a specific time.

_8164_set_fa_speed:

Sets the FA speed in home mode. If the FA speed is not set before home move, it will use 1/2 starting velocity as FA speed. The FA speed is also used in backlash compensation.

@ Syntax

C/C++ (DOS, Windows 95/NT/2K/XP)

```
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 mode);
I16 _8164_suppress_vibration(I16 AxisNo, U16 T1,
     U16 T2);
I16 _8164_set_idle_pulse(I16 AxisNo, I16
     idl_pulse);
I16 _8164_dwell_move(I16 AxisNo, F64 miniSecond);
I16 _8164_set_fa_speed(I16 AxisNo, F64 FA_speed);
```

Visual Basic (Windows 95/NT/2K/XP)

```
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 Mode As
     Integer) As Integer
B_8164_suppress_vibration (ByVal AxisNo As
     Integer, ByVal ReserveTime As Integer, ByVal
     Mode As Integer) As Integer
B_8164_set_idle_pulse(ByVal AxisNo As Integer,
     ByVal idl_pulse As Integer);
B_8164_dwell_move(ByVal AxisNo As Integer, ByVal
     miniSecond As Double);
B_8164_set_FA_speed(ByVal AxisNo As Integer,
     ByVal FA_Speed As Double);
```

@ Argument

AxisNo: Designated axis number 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 units of pulse per second

MaxVel: Starting velocity of a velocity profile in units of pulse per second

Tacc: Specified acceleration time in units of seconds

Tdec: Specified deceleration time in units of seconds

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

pcs_logic: Specify he pcs logic.

- Value = 0: low active
- Value = 1: high active

Type: define the SD pin usage

- Value = 0 : SD pin as SD signal
- Value = 1: SD pin as PCS signal

BcompPulse: Specified number of corrective pulses, 12-bit

Mode: 0 turns off, 1: enable backlash compensation, 2: slip correction

T1: Specified Reverse Time, 0 - 65535, unit 1.6 us

T2: Specified Forward Time, 0 - 65535, unit 1.6 us

Idl_pulse: Idl_pulse=0 - 7

miniSecond: time of dwell move, the unit is in ms

FA_Speed: the speed of FA

@ 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 an 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 an 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 an 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 an 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 an absolute 2-axis linear interpolation for any 2 axes,, with trapezoidal profile

_8164_start_sa_line2 – Begin an 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 an absolute 3-axis linear interpolation with trapezoidal profile

_8164_start_sa_line3 – Begin an 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 an absolute 4-axis linear interpolation with trapezoidal profile

_8164_start_sa_line4 – Begin an absolute 4-axis linear interpolation with S-curve profile

_8164_set_axis_option – Choose the interpolation speed mode

@ Description

Function	No. of interpolating axes	Velocity Profile	Relative / Absolute	Target Axes
_8164_start_tr_move_xy	2	T	R	Axes 0 & 1
_8164_start_ta_move_xy	2	T	A	Axes 0 & 1
_8164_start_sr_move_xy	2	S	R	Axes 0 & 1
_8164_start_sa_move_xy	2	S	A	Axes 0 & 1
_8164_start_tr_move_zu	2	T	R	Axes 2 & 3
_8164_start_ta_move_zu	2	T	A	Axes 2 & 3
_8164_start_sr_move_zu	2	S	R	Axes 2 & 3
_8164_start_sa_move_zu	2	S	A	Axes 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/2K/XP)

```
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);
I16 FNTYPE _8164_set_axis_option(I16 AxisNo, I16
     option);
```

Visual Basic (Windows 95/NT/2K/XP)

```
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
B_8164_set_axis_option (ByVal AxisNo As Integer,
        ByVal option1 As Integer) As Integer
```

@ Argument

CardNo: Designated card number 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 units of pulse per second

MaxVel: starting velocity of a velocity profile in units of pulse per second

Tacc: specified acceleration time in units of seconds

Tdec: specified deceleration time in units of seconds

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.

Option1:

- 0=default line move mode
- 1=Composite speed constant mode

@ 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 an 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 an absolute circular interpolation for Z & U

_8164_start_r_arc2 – Begin a relative circular interpolation for any 2 axes

_8164_start_a_arc2 – Begin an absolute circular interpolation for any 2 axes

_8164_start_tr_arc_xyu – Begin a T-curve relative circular interpolation

_8164_start_ta_arc_xyu – Begin a T-curve absolute circular interpolation

_8164_start_sr_arc_xyu – Begin a S-curve relative circular interpolation

_8164_start_sa_arc_xyu – Begin a S-curve absolute circular interpolation

_8164_start_tr_arc_yzu – Begin a T-curve relative circular interpolation

_8164_start_ta_arc_yzu – Begin a T-curve absolute circular interpolation

_8164_start_sr_arc_yzu – Begin a S-curve relative circular interpolation

_8164_start_sa_arc_yzu – Begin a T-curve absolute circular interpolation

_8164_start_tr_arc2 – Begin a T-curve relative circular interpolation

_8164_start_ta_arc2 – Begin a T-curve absolute circular interpolation

_8164_start_sr_arc2 – Begin a S-curve relative circular interpolation

_8164_start_sa_arc2 – Begin a S-curve absolute circular interpolation

_8164_start_tr_arc_xy – Begin a T-curve relative circular interpolation

_8164_start_ta_arc_xy – Begin a T-curve absolute circular interpolation

_8164_start_tr_arc_zu – Begin a T-curve relative circular interpolation

_8164_start_ta_arc_zu – Begin a T-curve absolute circular interpolation

_8164_start_sr_arc_xy – Begin a S-curve relative circular interpolation

_8164_start_sa_arc_xy – Begin a S-curve absolute circular interpolation

_8164_start_sr_arc_zu – Begin a S-curve relative circular interpolation

_8164_start_sa_arc_zu – Begin a S-curve absolute circular interpolation

@ Description

Function	Relative / Absolute	Speed Profile	Target Axes	Hardware version bit 12
_8164_start_r_arc_xy	R	Flat	Axes 0 & 1	0 or 1
_8164_start_a_arc_xy	A	Flat	Axes 0 & 1	0 or 1
_8164_start_r_arc_zu	R	Flat	Axes 2 & 3	0 or 1
_8164_start_a_arc_zu	A	Flat	Axes 2 & 3	0 or 1
_8164_start_r_arc2	R	Flat	Any 2 of 4	0 or 1
_8164_start_a_arc2	A	Flat	Any 2 of 4	0 or 1
_8164_start_tr_arc_xyu	R	T-curve	Axes 0 & 1	0 or 1
_8164_start_ta_arc_xyu	A	T-Curve	Axes 0 & 1	0 or 1
_8164_start_sr_arc_xyu	R	S-Curve	Axes 1 & 2	0 or 1
_8164_start_sa_arc_xyu	A	S-Curve	Axes 1 & 2	0 or 1
_8164_start_tr_arc_xy	R	T-curve	Axes 0 & 1	1
_8164_start_ta_arc_xy	A	T-Curve	Axes 0 & 1	1
_8164_start_sr_arc_xy	R	S-Curve	Axes 0 & 1	1
_8164_start_sa_arc_xy	A	S-Curve	Axes 0 & 1	1
_8164_start_tr_arc_zu	R	T-curve	Axes 2 & 3	1
_8164_start_ta_arc_zu	A	T-Curve	Axes 2 & 3	1
_8164_start_sr_arc_zu	R	S-Curve	Axes 2 & 3	1
_8164_start_sa_arc_zu	A	S-Curve	Axes 2 & 3	1
_8164_start_tr_arc2	R	T-curve	Any 2 of 4	1
_8164_start_ta_arc2	A	T-Curve	Any 2 of 4	1
_8164_start_sr_arc2	R	S-Curve	Any 2 of 4	1
_8164_start_sa_arc2	A	S-Curve	Any 2 of 4	1

@ Syntax

C/C++ (DOS, Windows 95/NT/2K/XP)

```
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);
I16 _8164_start_tr_arc_xyu(I16 CardNo, F64
    OffsetCx, F64 OffsetCy, F64 OffsetEx, F64
    OffsetEy, I16 DIR, F64 StrVel, F64 MaxVel,
    F64 Tacc);
I16 _8164_start_ta_arc_xyu(I16 CardNo, F64 Cx,
    F64 Cy, F64 Ex, F64 Ey, I16 DIR, F64 StrVel,
    F64 MaxVel, F64 Tacc);
I16 _8164_start_sr_arc_xyu(I16 CardNo, F64
    OffsetCx, F64 OffsetCy, F64 OffsetEx, F64
    OffsetEy, I16 DIR, F64 StrVel, F64 MaxVel,
    F64 Tacc, F64 SVacc);
I16 _8164_start_sa_arc_xyu(I16 CardNo, F64 Cx,
    F64 Cy, F64 Ex, F64 Ey, I16 DIR, F64 StrVel,
    F64 MaxVel, F64 Tacc, F64 SVacc);
I16 _8164_start_tr_arc_yzu(I16 CardNo, F64
    OffsetCx, F64 OffsetCy, F64 OffsetEx, F64
    OffsetEy, I16 DIR, F64 StrVel, F64 MaxVel,
    F64 Tacc);
I16 _8164_start_ta_arc_yzu(I16 CardNo, F64 Cx,
    F64 Cy, F64 Ex, F64 Ey, I16 DIR, F64 StrVel,
    F64 MaxVel, F64 Tacc);
I16 _8164_start_sr_arc_yzu(I16 CardNo, F64
    OffsetCx, F64 OffsetCy, F64 OffsetEx, F64
    OffsetEy, I16 DIR, F64 StrVel, F64 MaxVel,
    F64 Tacc, F64 SVacc);
I16 _8164_start_sa_arc_yzu(I16 CardNo, F64 Cx,
    F64 Cy, F64 Ex, F64 Ey, I16 DIR, F64 StrVel,
    F64 MaxVel, F64 Tacc, F64 SVacc);
I16 _8164_start_tr_arc2(I16 CardNo, I16
    *AxisArray, F64 OffsetCx, F64 OffsetCy, F64
    OffsetEx, F64 OffsetEy, I16 DIR, F64
    StrVel,F64 MaxVel, F64 Tacc,F64 Tdec);
I16 _8164_start_ta_arc2(I16 CardNo, I16
    *AxisArray, F64 Cx, F64 Cy, F64 Ex, F64 Ey,
    I16 DIR, F64 StrVel, F64 MaxVel, F64
    Tacc,F64 Tdec);
```

```
I16 _8164_start_sr_arc2(I16 CardNo, I16
    *AxisArray, F64 OffsetCx, F64 OffsetCy, F64
    OffsetEx, F64 OffsetEy, I16 DIR, F64
    StrVel,F64 MaxVel, F64 Tacc,F64 Tdec,F64
    SVacc,F64 SVdec);
I16 _8164_start_sa_arc2(I16 CardNo, I16
    *AxisArray, F64 Cx, F64 Cy, F64 Ex, F64 Ey,
    I16 DIR, F64 StrVel, F64 MaxVel, F64
    Tacc,F64 Tdec,F64 SVacc,F64 SVdec);
I16 _8164_start_tr_arc_xy(I16 CardNo, F64
    OffsetCx, F64 OffsetCy, F64 OffsetEx, F64
    OffsetEy, I16 DIR, F64 StrVel,F64 MaxVel,F64
    Tacc,F64 Tdec);
I16 _8164_start_ta_arc_xy(I16 CardNo, F64 Cx, F64
    Cy, F64 Ex, F64 Ey, I16 DIR, F64 StrVel,F64
    MaxVel,F64 Tacc,F64 Tdec);
I16 _8164_start_tr_arc_zu(I16 CardNo, F64
    OffsetCx, F64 OffsetCy, F64 OffsetEx, F64
    OffsetEy, I16 DIR, F64 StrVel,F64 MaxVel,F64
    Tacc,F64 Tdec);
I16 _8164_start_ta_arc_zu(I16 CardNo, F64 Cx, F64
    Cy, F64 Ex, F64 Ey, I16 DIR, F64 StrVel,F64
    MaxVel,F64 Tacc,F64 Tdec);
I16 _8164_start_sr_arc_xy(I16 CardNo, F64
    OffsetCx, F64 OffsetCy, F64 OffsetEx, F64
    OffsetEy, I16 DIR, F64 StrVel,F64 MaxVel,
    F64 Tacc,F64 Tdec,F64 SVacc,F64 SVdec);
I16 _8164_start_sa_arc_xy(I16 CardNo, F64 Cx, F64
    Cy, F64 Ex, F64 Ey, I16 DIR, F64 StrVel,F64
    MaxVel, F64 Tacc,F64 Tdec,F64 SVacc,F64
    SVdec);
I16 _8164_start_sr_arc_zu(I16 CardNo, F64
    OffsetCx, F64 OffsetCy, F64 OffsetEx, F64
    OffsetEy, I16 DIR, F64 StrVel,F64 MaxVel,
    F64 Tacc,F64 Tdec,F64 SVacc,F64 SVdec);
I16 _8164_start_sa_arc_zu(I16 CardNo, F64 Cx, F64
    Cy, F64 Ex, F64 Ey, I16 DIR, F64 StrVel,F64
    MaxVel, F64 Tacc,F64 Tdec,F64 SVacc,F64
    SVdec);
```

Visual Basic (Windows 95/NT/2K/XP)

```
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
B_8164_start_tr_arc_xyu (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 StrVel As Double, ByVal MaxVel As
      Double, ByVal Tacc As Double) As Integer
B_8164_start_ta_arc_xyu (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 StrVel As
```

```
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double) As Integer
B_8164_start_sr_arc_xyu (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 StrVel As Double, ByVal MaxVel As
        Double, ByVal Tacc As Double) As Integer
B_8164_start_sa_arc_xyu (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 StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal SVacc As Double) As Integer
B_8164_start_tr_arc_yzu (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 StrVel As Double, ByVal MaxVel As
        Double, ByVal Tacc As Double) As Integer
B_8164_start_ta_arc_yzu (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 StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double) As Integer
B_8164_start_sr_arc_yzu (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 StrVel As Double, ByVal MaxVel As
        Double, ByVal Tacc As Double) As Integer
B_8164_start_sa_arc_yzu (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 StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal SVacc As Double) As Integer
B_8164_start_tr_arc2 (ByVal CardNo As Integer,
        AxisArray As Double, ByVal OffsetCx As
        Double, ByVal OffsetCy As Double, ByVal
        OffsetEx As Double, ByVal OffsetEy As
        Double, ByVal DIR As Integer, ByVal StrVel
```

```
        As Double, ByVal MaxVel As Double, ByVal
        Tacc As Double, ByVal Tdec As Double) As
        Integer
B_8164_start_ta_arc2 (ByVal CardNo As Integer,
        AxisArray As Double, ByVal Cx As Double,
        ByVal Cy As Double, ByVal Ex As Double,
        ByVal Ey As Double, ByVal DIR As Integer,
        ByVal StrVel As Double, ByVal MaxVel As
        Double, ByVal Tacc As Double, ByVal Tdec As
        Double) As Integer
B_8164_start_sr_arc2 (ByVal CardNo As Integer,
        AxisArray As Double, ByVal OffsetCx As
        Double, ByVal OffsetCy As Double, ByVal
        OffsetEx As Double, ByVal OffsetEy As
        Double, ByVal DIR As Integer, 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_arc2 (ByVal CardNo As Integer,
        AxisArray As Double, ByVal Cx As Double,
        ByVal Cy As Double, ByVal Ex As Double,
        ByVal Ey As Double, ByVal DIR As Integer,
        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_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 StrVel As Double, ByVal MaxVel As
        Double, ByVal Tacc As Double, ByVal Tdec As
        Double) As Integer
B_8164_start_ta_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 StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double) As Integer
B_8164_start_tr_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 StrVel As Double, ByVal MaxVel As
        Double, ByVal Tacc As Double, ByVal Tdec As
        Double) As Integer
B_8164_start_ta_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 StrVel As
        Double, ByVal MaxVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double) As Integer
B_8164_start_sr_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 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_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 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_sr_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 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_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 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: Designated card number 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 center

Ex: Specified X-axis absolute position end of arc

Ey: Specified Y-axis absolute position end of arc

DIR: Specified direction of arc, CW:0 , CCW:1

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[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.

_8164_escape_home – Escape home

_8164_home_search – Perform auto home search

@ Description

_8164_set_home_config:

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

_8164_home_move:

Causes the axis to perform a home return move according to the **_8164_set_home_config()** function settings. The direction of movement is determined by the sign of velocity parameter (svel, mvel). Since the stopping condition of this function is determined by the *home_mode* setting, you must take caution in selecting the initial moving direction. You must also take caution when handling conditions when the limit switch is touched or other conditions that are possible causing the axis to stop. Executing **v_stop()** function during **home_move()** can also cause the axis to stop.

_8164_escape_home:

After homing, this function leaves the home switch.

_8164_home_search:

Starts home searching regardless of the location of axis. The ORGoffset must be set to non-zero to previous miss operation.

@ Syntax

C/C++ (DOS, Windows 95/NT/2K/XP)

```
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);
I16 _8164_escape_home(I16 AxisNo, F64 SrVel,F64
     MaxVel,F64 Tacc);
I16 _8164_home_search(I16 AxisNo, F64 StrVel, F64
     MaxVel, F64 Tacc, F64 ORGOffset);
```

Visual Basic (Windows 95/NT/2K/XP)

```
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
B_8164_escape_home(ByVal AxisNo As Integer, ByVal
     SrVel As Double, ByVal MaxVel As Double,
     ByVal Tacc As Double);
B_8164_home_search (ByVal AxisNo As Integer,
     ByVal StrVel As Double, ByVal MaxVel As
     Double, ByVal Tacc As Double, ByVal
     ORGOffset As Double) As Integer
```

@ Argument

AxisNo: Designated axis number to configure and perform home return

home_mode: Stopping modes for home return, 0-12

- (Please refer to section 4.1.8)

org_logic: Action logic configuration for ORG

- **org_logic**=0, active low
- **org_logic**=1, active high

EZ_logic: Action logic configuration for E

- **EZ_logic**=0, active low
- **EZ_logic**=1, active high

ez_count: 0-15 (Refer to section 4.1.8)

erc_out: Set ERC output options

- **erc_out** =0, no erc out;
- **erc_out** =1, erc out when home finishing

StrVel: starting velocity of a velocity profile in units of pulse per second

MaxVel: starting velocity of a velocity profile in units of pulse per second

Tacc: specified acceleration time in units of seconds

ORGOffset: The escape pulse amounts when home search touches the ORG singal

@ 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

_8164_set_pulser_ratio – Set manual pulser ratio for actual output pulse rate

_8164_pulser_r_line2 – Pulser mode for 2-axis linear interpolation

_8164_pulser_r_arc2 – Pulser mode for 2-axis arc interpolation

@ Description

_8164_set_pulser_iptmode:

Configures the input mode of manual pulser.

_8164_pulser_vmove:

With this command, the axis begins to move according to the manual pulse input. The axis outputs one pulse when it receives one manual pulse, until the **sd_stop** or **emg_stop** command is written.

_8164_pulser_pmove:

With this command, the axis begins to move according to the manual pulse input. The axis outputs one pulse when it receives one manual pulse, until the **sd_stop** or **emg_stop** command is written or the output pulse number reaches the distance.

_8164_pulser_home_move:

With this command, the axis begins to move according to manual pulse input. The axis outputs one pulse when it receives one manual pulse, until the **sd_stop** or **emg_stop** command is written or the home move finishes.

_8164_set_pulser_ratio:

Sets the manual pulse ratio for actual output pulse rate. The formula for manual pulse output rate is:

- Output Pulse Count = Input Pulse Count * (PMG+1) * [(PDV+1)/2048]
- The PDV=0-2047 Divide Factor
- The PMG=0-31 Multiplication Factor

_8164_pulser_r_line2:

Pulser mode for 2-axis linear interpolation (relative mode only).

_8164_pulser_r_arc2:

Pulser mode for 2-axis arc interpolation (relative mode only)

@ Syntax

C/C++ (DOS, Windows 95/NT/2K/XP)

```
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);
I16 _8164_set_pulser_ratio(I16 AxisNo,I16 PDV,
     I16 PMG);
I16 _8164_pulser_r_line2(I16 CardNo,I16
     *AxisArray, F64 DistX, F64 DistY, F64
     SpeedLimit);
I16 _8164_pulser_r_arc2(I16 CardNo, I16
     *AxisArray, F64 OffsetCx, F64 OffsetCy, F64
     OffsetEx, F64 OffsetEy, I16 DIR, F64
     MaxVel);
```

Visual Basic (Windows 95/NT/2K/XP)

```
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
B_8164_set_pulser_ratio(ByVal AxisNo As Integer,
     ByVal PDV As Integer, ByVal PMG As Integer)
     As Integer
B_8164_pulser_r_line2(ByVal CardNo As Integer,
     AxisArray As Integer, ByVal DistX As Double,
     ByVal DistY As Double, ByVal SpeedLimit As
     Double) As Integer
B_8164_pulser_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

`AxisNo`: Designated axis number to start manual move

`InputMode`: Setting of manual pulse input mode from the 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 a manual pulse move

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

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.

PDV, **PMG**: Divide and Multi Factor

- PDV=0-2047 Divide Factor
- PMG=0-31 Multi Factor

Output Pulse Count = Input Pulse Count * (PMG+1) * [(PDV+1)/2048]

DistX: specified relative distance of axis 0 to move

DistY: specified relative distance of axis 1 to move

OffsetCx: X-axis offset from center

OffsetCy: Y-axis offset from center

OffsetEx: X-axis offset from end of arc

OffsetEy: Y-axis offset from end of arc

DIR: Specified direction of arc, CW:0 , CCW:1

SpeedLimit: Maximum tangential velocity in units of pulse per second

MaxVel: Maximum tangential velocity in units of pulse per second

@ Return Code

```
ERR_NoError
ERR_PulserHomeTypeError
```

6.11 Motion Status

@ Name

_8164_motion_done – Return the motion status

@ Description

_8164_motion_done:

Returns the motion status of the 8164. If the card is in stopped condition, check if it is in error stop status by reading error interrupt status in section 6.14.

@ Syntax

C/C++ (DOS, Windows 95/NT/2K/XP)

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

Visual Basic (Windows 95/NT/2K/XP)

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

@ Argument

AxisNo: Designated axis number to start manual move

@ Return Value

0	Under Stopped condition
1	Reserved
2	Wait CSTA (Synchronous start signal)
3	Wait Internal sync. signal
4	Wait for another axis to stop
5	Wait ERC timer finished
6	Wait DIR Change timer finished
7	Backlash compensating
8	Wait PA/PB input
9	In FA speed motion
10	In start velocity motion
11	In acceleration
12	In Max velocity motion
13	In deceleration
14	Wait INP input
15	Others

6.12 Motion Interface I/O

@ Name

_8164_set_alm – Set alarm logic and operating mode

_8164_set_el – Set EL stopping 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:

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

_8164_set_el:

Sets the stopping mode of the EL active.

_8164_set_inp:

Sets the active logic of the In-Position signal input from the servo driver. You can select whether to enable or disable this function. The function is disabled by default.

_8164_set_erc:

Sets the logic and on time of the ERC.

_8164_set_servo:

Sets the ON/OFF state of the SVON signal. The default value is 1 (OFF), which means that the SVON is open to GND.

_8164_set_sd:

Sets the active logic, latch control, and operating mode of the SD signal input from a mechanical system. You can select whether to enable or disable this function. This function is disabled by default.

@ Syntax

C/C++ (DOS, Windows 95/NT/2K/XP)

```
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/2K/XP)

```
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`: Designated axis number to configure

`alm_logic`: Setting of active logic for ALARM signals

- **`alm_logic`**=0, active LOW
- **`alm_logic`**=1, active HIGH

alm_mode: Reacting modes when receiving an ALARM signal

- **alm_mode**=0, motor immediately stops (default)
- **alm_mode**=1, motor decelerates then stops.

el_mode: Reacting modes when receiving an EL signal

- **el_mode**=0, motor immediately stops (default)
- **el_mode**=1, motor decelerates then stops.

inp_enable: INP function enabled/disabled

- **inp_enable**=0, Disabled (default)
- **inp_enable**=1, Enabled

inp_logic: Set the active logic for the INP signal

- **inp_logic**=0, active LOW
- **inp_logic**=1, active HIGH

erc_logic: Set the active logic for the ERC signal

- **erc_logic**=0, active LOW
- **erc_logic**=1, active HIGH

erc_on_time: Set the time length of ERC active

- **erc_on_time**=0, 12μs
- **erc_on_time**=1, 102μs
- **erc_on_time**=2, 409μs
- **erc_on_time**=3, 1.6ms
- **erc_on_time**=4, 13ms
- **erc_on_time**=5, 52ms
- **erc_on_time**=6, 104ms
- **erc_on_time**=7, Level output

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: Set the active logic for the SD signal

- **sd_logic**=0, active LOW
- **sd_logic**=1, active HIGH

sd_latch: Set the latch control for the SD signal

- **sd_latch**=0, do not latch
- **sd_latch**=1, latch

sd_mode: Set the reacting mode of the SD signal

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

el_logic: Set EL as normal low (0) or normal high(1)

@ Return Code

```
ERR_NoError
```

6.13 Motion I/O Monitoring

@ Name

_8164_get_io_status – Get all the motion I/O statuses of each 8164

@ Description

_8164_get_io_status:

Obtains all the I/O statuses for each axis. The definition for each bit is as follows:

Bit	Name	Description
0	RDY	RDY pin input
1	ALM	Alarm Signal
2	+EL	Positive Limit Switch
3	-EL	Negative Limit Switch
4	ORG	Origin Switch
5	DIR	DIR output
6	EMG	EMG status
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/2K/XP)

```
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. “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)

_8164_set_axis_stop_int – enable axis stop int

_8164_mask_axis_stop_int – mask axis stop int

@ Description

_8164_int_control:

Enables interrupt generating to host computer.

_8164_set_int_factor:

Allows factor selection to initiate the event int. The error can never be masked once the interrupt service is turned on by **_8164_int_control()**.

The INT status of 8164 is composed of two independent parts: **error_int_status** and **event_int_status**. The **event_int_status** recordes 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 the 8164, for example: EL, ALM …etc. This kind of INT cannot be masked. Below is the definition of these two int_status. By setting the relative bit as 1, the card can generate INT signal to the host computer.

Interrupt factors

Bit	Description
0	Normal Stop
1	Next command Starts
2	Command pre-register 2 is empty
3	(Reserved)
4	Acceleration Start
5	Acceleration End
6	Deceleration Start
7	Deceleration End
8	Positive soft limit on
9	Negative soft limit on
10	Comparator 3 compared
11	General Comparator compared
12	Trigger Comparator compared
13	(Reserved)
14	Counter Latched for axis2,3
15	ORG Latch Active
16	SD on
17	(Reserved)
18	(Reserved)
19	CSTA, Sync. Start on
20-31	(Reserved)

_8164_int_enable: (For Window only.)

Enables the Windows INT event.

_8164_int_disable: (For Window only.)

Disables the Windows INT event.

_8164_get_int_status: (For Window only.)

Identifies the cause of the interrupt signal. After the value is obtained, the status register will be cleared to 0. The return value is two, 32-bit 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	+Soft Limit on and stop
1	-Soft Limit on and stop
2	(Reserved)
3	General Comparator on and Stop
4	(Reserved)
5	+End Limit on and stop
6	-End Limit on and stop
7	ALM happen and stop
8	CSTP, Sync. Stop on and stop
9	CEMG, Emergency on and stop
10	SD on and slow down to stop
11	(Reserved)
12	Interpolation Error and stop
13	Other Axis stop on Interpolation
14	Pulser input buffer overflow and stop
15	Interpolation counter overflow
16	Encoder input signal error
17	Pulser input signal error
11-31	(Reserved)

The second is for event_int_status, which can be masked by **_8164_set_int_factor()**. See table below:

event_int_status: can be masked by function call **_8164_int_factor()**

Bit	Description
0	Normal Stop
1	Next command Starts
2	Command pre-register 2 is empty
3	(Reserved)
4	Acceleration Start
5	Acceleration End
6	Deceleration Start
7	Deceleration End
8	(Reserved)
9	(Reserved)
10	(Reserved)
11	General Comparator compared
12	Trigger Comparator compared
13	(Reserved)
14	Counter Latched for axes 2 and 3
15	ORG Input and Latched
16	SD on
17	(Reserved)
18	(Reserved)
19	CSTA, Sync. Start on
20-30	(Reserved)
31	Axis Stop Interrupt

_8164_link_interrupt: (Windows only.)

Links to the interrupt call back function.

_8164_get_int_type: (DOS only)

Detects the type of INT that occurred.

_8164_enter_isr: (DOS only)

This function is used to inform the system that the process is now entering interrupt service routine.

_8164_leave_isr: (DOS only)

This function is used to inform the system that the process is now leaving interrupt service routine.

_8164_get_event_int: (DOS only)

This function is used to get event_int_status.

_8164_get_error_int: (DOS only)

This function is used to get error_int_status.

_8164_get_irq_status: (DOS only)

This function allows user to confirm if the designated card generates the INT signal to host PC.

_8164_not_my_irq: (DOS only)

This function must be called after the designated card generates the INT signal to host PC.

_8164_isr0, **_8164_isr1**, **_8164_isr2**, **_8164_isr3**, **…..** **_8164_isr9**, **_8164_isra**, **_8164_isrb**: **(DOS only)**

Individual Interrupt service routine for cards 0-11.

_8164_set_axis_stop_int:

Enables an axis stop interrupt. Once it is enabled, the interrupt happens no matter if it is a normal stop or error stop. This interrupt condition can be turned on or off accompanied with every motion command by setting **_8164_mask_axis_stop_int()**. This kind of interrupt condition is different from **_8164_set_int_factor()**. It can be controlled in each motion command, which is very useful in continuous motion when you only need a final command interrupt. The interrupt status is in “event interrupt status” bit 31.

_8164_mask_axis_stop_int:

This function will affect the axis stop interrupt factor which is set by **_8164_set_axis_stop_int()**.

@ 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));
I16 _8164_set_axis_stop_int(I16 AxisNo, I16
     axis_stop_int);
I16 _8164_mask_axis_stop_int(I16 AxisNo, I16
     int_mask);
```

Visual Basic (Windows 95/NT/2K/XP)

```
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
B_8164_mask_axis_stop_int (ByVal AxisNo As
     Integer, ByVal int_mask As Integer) As
     Integer
B_8164_set_axis_stop_int (ByVal AxisNo As
     Integer, ByVal axis_stop_int As Integer) 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

int_mask: (0:make axis stop interrupt active, 1:make axis stop interrupt in-active)

axis_stop_int: (0: disable axis stop interrupt factor, 1: enable axis stop interrupt factor)

@ 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():

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

_8164_set_position():

Changes the feedback position counter to the specified value. Note that the value to be set will be processed by the move ratio. If move ratio is 0.5, then the set value will be twice as the given value.

_8164_get_command():

Reads the value of the command position counter. The source of the command position counter is the pulse output of the 8164.

_8164_set_command():

Changes the value of the command position counter.

_8164_get_error_counter():

Reads the value of the position error counter.

_8164_reset_error_counter():

Clears the position error counter.

_8164_get_general_counter():

Read the value of the general counter.

_8164_set_general_counter():

Sets the counting source of and change the value of general counter. By default, the source is pulse input.

_8164_get_target_pos():

Reads the value of the target position recorder. The target position recorder is maintained by the card software driver. It records the position to settle down for current running motion.

_8164_reset_target_pos():

Sets new value for the target position recorder. It is necessary to call this function when home return completes or when a new feedback counter value is set by function **_8164_set_position()**.

_8164_get_rest_command():

Reads the remaining pulse counts until the end of the current motion.

_8164_check_rdp():

Reads the ramping down point data. The ramping down point is the position where deceleration starts. The data is stored as a pulse count and it causes 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/2K/XP)

```
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

6.15.1 @ 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

_8164_force_cmp_output – Force to output trigger pulses from CMP

_8164_set_compare_mode – A general function for setting comparator **mode**

_8164_set_compare_data – A general function for setting comparator data

_8164_set_rotary_counter – set the counter as a rotary counter

@ Description

_8164_set_ltc_logic:

Sets the logic of the latch input. This function is applicable only for last two axes in every 8164 card.

_8164_get_latch_data:

After the latch signal arrived, this function reads the latched value of counters.

_8164_set_soft_limit:

Sets the soft limit value .

_8164_enable_soft_limit, **_8164_disable_soft_limit**:

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:

Enables the step losing checking facility. By giving a tolerance value, the card generates an interrupt (event_int_status , bit 10) when position error counter exceed tolerance.

_8164_set_general_comparator:

Sets the source and comparing value for the general comparator. When the source counter value reached the comparing value, the 8164 will generate an interrupt (event_int_status , bit 11).

_8164_set_trigger_comparator:

Sets the comparing method and value for the trigger comparator. When the feedback position counter value reaches the comparing value, the card generates a trigger and a pulse output via **CMP** and an interrupt (event_int_status, bit 12) is sent to the host computer. If _8164_set_auto_compare is used, then the comparing value set by this function is ignored automatically.

Note: it is applicable only for first two axes in every 8164 card.

_8164_set_trigger_type:

Sets the trigger output mode

In hardware version A2, it is used for setting the output pulse as a one shot or constant on.

In hardware version A3, it is used for setting the output pulse as normal high or normal low.

_8164_check_compare_data:

Obtains the current comparing data of the designated comparator.

_8164_check_compare_status:

Obtains the status of all comparators. When some comparators come into existence, the relative bits of cmp_sts will become '1,' otherwise **0**.

_8164_set_auto_compare:

Sets the comparing data source of the trigger comparator. The source can be either a function or a table.

_8164_build_compare_function:

Builds a comparing function by defining the start/end point and interval. There is no limitation on the max number of comparing data. It automatically loads a final point after user's end point. That is, (end point + Interval x total points) x move ratio.

Note: Turn off all interrupt functions when triggering is running.

_8164_build_compare_table:

Builds a comparing table by defining a data array. The size of array is limited to 1024 when using RAM mode.

Note: Turn off all interrupt functions when triggering is running.

_8164_cmp_v_change:

Sets up the comparator velocity change function. It is a v_change function but acts when a general comparator comes into existence. When this function is issued, the parameter "CmpAction" of **_8164_set_general_comparator()** must be set to **3** which is for speed change option.

The compare data is also set by **_8164_set_general_comparator()**, while the remain distance, compare point's velocity, new velocity, and acceleration time are set by **_8164_cmp_v_change()**.

_8164_force_cmp_output:

Ouputs trigger pulses from CMP without comparing position. Meanwhile, only axis 0 and axis 1 can use this function.

_8164_set_rotary_counter:

Sets the counter of axis as a rotary (ring) counter. Once it is set as a rotary counter, the value will be reset upon the condition matched.

@ 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 LatchNo,
     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
     CmpSrc,  I16 CmpMethod, F64 Data);
I16 _8164_set_trigger_type(I16 AxisNo, I16
     TriggerType);
I16 _8164_check_compare_data(I16 AxisNo, I16
     CompType, 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)
I16 _8164_force_comp_output(I16 AxisNo);
I16 _8164_set_compare_mode(I16 AxisNo, I16
     CompNo, I16 CmpSrc, I16 CmpMethod, I16
     CmpAction);
I16 _8164_set_compare_data(I16 AxisNo, I16
     CompNo, F64 Pos);
I16 _8164_set_rotary_counter(I16 AxisNo ,I16
     reset_src);
```

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, I32 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, I32 Size);
```

Visual Basic (Windows 95/NT/2K/XP)

```
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 CmpSrc 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 CompType 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)
B_8164_force_cmp_output (ByVal AxisNo As Integer)
     As Integer
B_8164_set_compare_mode(ByVal AxisNo As Integer,
     ByVal CompNo As Integer, ByVal CmpSrc As
     Integer, ByVal CmpMethod As Integer, ByVal
     CmpAction As Integer) As Integer
B_8164_set_compare_data (ByVal AxisNo As Integer,
     ByVal CompNo Integer, ByVal Pos As Double)
     As Integer
B_8164_set_rotary_counter (ByVal AxisNo As
     Integer, ByVal reset_src As Integer) As
     Integer
```

@ 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

LatchNo: Specified Counter to latch

- LatchNo = 1 , Command counter
- LatchNo = 2 , Feedback counter
- LatchNo = 3 , Error Counter
- LatchNo = 4 , General Counter

Pos: Latched counter value,

PLimit: Soft limit value, positive direction

NLimit: Soft limit value, 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 form 0 - 32767

On_Off: Enable/Disable step-losing detection

- On_Off =0, Disable
- On_Off =1, Enable

CompNo: The comparator number of axis

- CompNo =1, Comparator1, default for positive soft limit
- CompNo =2, Comparator2, default for negative soft limit
- CompNo =3, Comparator3, default for for position error check
- CompNo =4, Comparator4, It could be any counter source
- CompNo =5, Comparator5, default for triggering output

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
- CmpMethod =6, Reserved

CmpAction: The reacting mode when comparison comes into exist

- CmpAction =0, INT and Internal sync. signal output
- CmpAction =1, Immediate stop
- CmpAction =2, Slow down then stop
- CmpAction =3, Speed change

Data: Comparing value

TriggerType: Selection of type of trigger output mode

- Hardware Version A2
  - TriggerType =0, one shoot (default)
  - TriggerType =1, constant high
- Hardware Version A3
  - TriggerType =0, normal high (default)
  - TriggerType =1, normal low

CompType: Selection of type of comparator

- CompType =1, + Soft Limit
- CompType =2, -Soft Limit
- CompType =3, Error Counter Comparator Value
- CompType =4, General Comparator Value
- CompType =5, Trigger Output Comparator Value

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, compare function on RAM
- SelectSrc =3, compare table on RAM

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 for axis2,3 of card, outputting pulses to DO0,1
- Device =1, FIFO for axis 0,1 of card, outputting pulses to CMP0,1

Res_dist: The remaining distance from the compare point. After comparison, the original target position is ignored, and the axis will keep moving the Res_dist.

oldvel: Velocity at compare point. Must be specified manually.

newvel: New velocity

AccTime: Acceleration time

reset_src: Reset source selection

- Value=0, disable
- Value=1, Latch pin ON
- Value=2, ORG pin ON
- Value=3, Comparator 4 ON
- Value=4, Comparator 5 ON

@ 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 – toggle continuous motion sequence flags

_8164_check_continuous_buffer – check if the command register buffer is empty

@ Description

_8164_set_continuous_move():

This function is needed before and after continuous motion command sequences.

_8164_check_continuous_buffer():

Detects whether the command pre-register is empty or not. Once the command pre-register is empty, you may write the next motion command into it. Otherwise, the new command overwrites the previous command in the 2nd command pre-register.

@ Syntax

C/C++ (DOS, Windows 95/NT/2K/XP)

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

Visual Basic (Windows 95/NT/2K/XP)

```
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`: Designated axis number

`conti_flag`: Flag for continuous motion

- conti_flag = 0, declare continuous motion sequence is finished
- conti_flag = 1, declare continuous motion sequence is started

@ Return Value

`ERR_NoError`

Return value of _8164_check_continuous_buffer():

- Hardware version bit 12=0
  - 0: command register 2 is empty
  - 1: command register 2 is in-use

Return value of _8164_check_continuous_buffer():

- Hardware version bit 12=1
  - 0: all command registers are empty
  - 1: command register is in-use
  - 2: command register 1 is in-use
  - 3: command register 2 is in-use

6.18 Multiple Axes Simultaneous Operation

@ Name

_8164_set_tr_move_all – Multi-axis simultaneous operation setup.

_8164_set_ta_move_all – Multi-axis simultaneous operation setup.

_8164_set_sr_move_all – Multi-axis simultaneous operation setup.

_8164_set_sa_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

_8164_set_sync_option – Other sync. motion setting

_8164_set_sync_stop_mode – Setting the stop mode of CSTOP signal

_8164_set_sync_signal_source – Synchronous signal source setting

_8164_set_sync_signal_mode – Synchronous signal mode setting

@ Description

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

When properly setup with **_8164_set_xx_move_all()**, the function **_8164_start_move_all()** causes all specified axes to begin a trapezoidal relative movement, and **_8164_stop_move_all()** stops them. Both functions guarantee that motion Start/Stop on all specified axes occur at the same time.

Note that proper CN4 connections are needed for these two functions. Refer to Section 3.14.

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

```
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]);

returnERR_NoError;
}
```

_8164_set_sync_option():

Allows two or more different command groups start at the same time. For example, if you want a 2-axis linear interpolation and a 1-

axis single motion to start at the same time, you can turn on this option before the command starts. This function may also be used when waiting for another command's finish signal before starting. For example, axis1 must start after axis2 is done.

_8164_set_sync_stop_mode():

Provides two options for stop types: immediately stop and slow down to stop. When the _8164_stop_move_all() or CSTOP signal is used, the axes stop according to this setting.

_8164_set_sync_signal_source:

Selects the synchrous signal source of one axis. It could be from itself or from the other three axes.

_8164_set_sync_signal_mode:

Specifies the internal synchronous signal output timing. If one of the condition is satisfied, the internal synchronous signal is triggered.

@ Syntax

C/C++ (DOS, Windows 95/NT/2K/XP)

```
I16 _8164_set_tr_move_all(I16 TotalAxes, I16
     *AxisArray, F64 *DistA, F64 *StrVelA, F64
     *MaxVelA, F64 *TaccA, F64 *TdecA);
I16 _8164_set_sa_move_all(I16 TotalAx, I16
     *AxisArray, F64 *PosA, F64 *StrVelA, F64
     *MaxVelA, F64 *TaccA, F64 *TdecA, F64
     *SVaccA, F64 *SVdecA);
I16 _8164_set_ta_move_all(I16 TotalAx, I16
     *AxisArray, F64 *PosA, F64 *StrVelA, F64
     *MaxVelA, F64 *TaccA, F64 *TdecA);
I16 _8164_set_sr_move_all(I16 TotalAx, I16
     *AxisArray, F64 *DistA, F64 *StrVelA, F64
     *MaxVelA, F64 *TaccA, F64 *TdecA, F64
     *SVaccA, F64 *SVdecA);
I16 _8164_start_move_all(I16 FirstAxisNo);
I16 _8164_stop_move_all(I16 FirstAxisNo);
I16 _8164_set_sync_option(I16 AxisNo, I16
     sync_stop_on, I16 cstop_output_on, I16
     sync_option1, I16 sync_option2);
```

```
I16 _8164_set_sync_stop_mode(I16 AxisNo, I16
     stop_mode);
I16 _8164_set_sync_signal_source(I16 AxisNo, I16
     Sync_axis);
I16 _8164_set_sync_signal_mode(I16 AxisNo, I16
     mode);
```

Visual Basic (Windows 95/NT/2K/XP)

```
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_set_sa_move_all(ByVal TotalAxes As
     Integer, AxisArray As Integer, PosA As
     Double, StrVelA As double, MaxVelA As
     double, TaccA As double, TdecA As double,
     SVaccA As double, SVdecA As Double);
B_8164_set_ta_move_all(ByVal TotalAxes As
     Integer, AxisArray As Integer, PosA As
     Double, StrVelA As double, MaxVelA As
     double, TaccA As double, TdecA As double);
B_8164_set_sr_move_all(ByVal TotalAxes As
     Integer, AxisArray As Integer, DistA As
     Double, StrVelA As double, MaxVelA As
     double, TaccA As double, TdecA As double,
     SVaccA As double, SVdecA As Double);
B_8164_start_move_all(ByVal FirstAxisNo  As
     Integer);
B_8164_stop_move_all(ByVal FirstAxisNo As
     Integer);
B_8164_set_sync_option (ByVal AxisNo As Integer,
     ByVal sync_stop_on As Integer, ByVal
     cstop_output_on As Integer, ByVal
     sync_option1 As Integer, ByVal sync_option2
     As Integer) As Integer
B_8164_set_sync_stop_mode (ByVal AxisNo As
     Integer, ByVal stop_mode As Integer) As
     Integer
B_8164_set_sync_signal_source (ByVal AxisNo As
     Integer, ByVal Sync_axis As Integer) As
     Integer
B_8164_set_sync_signal_mode (ByVal AxisNo As
     Integer, ByVal mode As Integer) 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 units of pulse

***StrVelA**: Starting velocity array in units of pulse per second

***MaxVelA**: Maximum velocity array in units of pulse per second

***TaccA**: Acceleration time array in units of seconds

***TdecA**: Deceleration time array in units of seconds

***SVaccA**: Specified velocity interval array in which S-curve acceleration is performed

***SVdecA**: specified velocity interval array in which S-curve deceleration is performed.

FirstAxisNo: the first element in AxisArray

Sync_stop_on: Axis will stop if the CSTOP signal is on

Cstop_output_on: CSTOP signal will output with an abnormal stop (ALM,EL..etc)

Sync_option1: Choose command start type:

- 0: default (immediately start)
- 1: waiting _8164_start_move_all() or CSTA signal
- 2: waiting Internal sync. signal to start (sync_source)
- 3: waiting Sync_option2's condition to start

Sync_option2: For example:

- 0: default (useless)
- 1: after Axis0 stops
- 2: after Axis1 stops
- 4: after Axis2 stops
- 8: after Axis3 stops
- 5: after Axis0 and Axis2 stop
- 15: Axis0-Axis3 stop

stop_mode:

- 0: immediate stop
- 1: slow down to stop

mode:

- 0=Off
- 1-5=compare 1-5 active
- 8=Acc start
- 9=Acc end
- 10=Dec start
- 11=Dec end

Sync_axis: 0-3, the axis index in the card.

@ Return Code

```
ERR_NoError
ERR_SpeedError
```

6.19 General-purposed TTL output (PCI-8164 Only)

@ Name

_8164_d_output – Digital Output

_8164_get_dio_status – Get DIO status

@ Description

_8164_d_output():

Sets the on_off status for general-purposed TTL digital output pin.

_8164_get_dio_status():

Reads the status of all digital output pins.

@ Syntax

C/C++ (DOS, Windows 95/NT/2K/XP)

```
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/2K/XP)

```
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
```

6.20 General-purposed DIO (MPC-8164/PXI-8164 only)

@ Name

_8164_write_do – Digital Output (MPC-8164 only)

_8164_read_di – Digital Input (MPC-8164 only)

_8164_write_axis_do – Digital Output (PXI-8164 only)

_8164_read_axis_di – Digital Input (PXI-8164 only)

@ Description

_8164_write_do():

Outputs an 8-bit value once to control eight output channels.

_8164_read_di():

Reads back an 8-bit value once from eight input channels.

_8164_write_axis_do():

Controls the axis' general digital output channel.

_8164_read_axis_di():

Reads back the axis' general digital input channel.

@ Syntax

C/C++ (DOS, Windows 95/NT/2K/XP)

```
I16 _8164_write_do(I16 CardNo, U16 Value);
U16 _8164_read_di(I16 CardNo);
I16 _8164_write_axis_do(I16 AxisNo, U16 Value);
U16 _8164_read_axis_di(I16 AxisNo);
```

Visual Basic (Windows 95/NT/2K/XP)

```
B_8164_write_do (ByVal CardNo As Integer, ByVal
     value As Integer) As Integer
B_8164_read_di (ByVal CardNo As Integer) As
     Integer
B_8164_write_axis_do (ByVal AxisNo As Integer,
     ByVal value As Integer) As Integer
B_8164_read_axis_di (ByVal AxisNo As Integer) As
     Integer
```

@ Argument

CardNo: Designated card number

AxisNo: Designated axis number

Value: Value for output

- Bit value =0, output OFF
- Bit value =1, output ON

@ Return Value

ERR_NoError

Digital Input Value for 8 channels in MPC-8164

Axis' general digital input value in PXI-8164

6.21 Card ID (PXI-8164 Only)

@ Name

_8164_enable_card_id – Enable card ID's function

_8164_check_card – Check if this Card ID exist

@ Description

_8164_enable_card_id():

Makes the card ID settings valid. This line must be placed before **_8164_initial()** function

_8164_check_card():

Checks if the card ID exists. You can use this function to search all the cards in the 0 - 11 range.

@ Syntax

C/C++ (DOS, Windows 95/NT/2K/XP)

```
void _8164_enable_card_id(void);
I16 _8164_check_card(I16 CardNo);
```

Visual Basic (Windows 95/NT/2K/XP)

```
B_8164_enable_card_id ()
B_8164_check_card (ByVal CardNo As Integer) As
     Integer
```

@ Argument

CardNo: Designated card number

@ Return Value

ERR_NoError

_8164_check_card():

- ▷ TRUE means card ID exist
- ▷ FALSE means card ID not exist

6.22 PXI Trigger Bus (PXI-8164 Only)

@ Name

_8164_get_pxi_trigger_value – Readback PXI_TRG's status value

_8164_set_pxi_trigger_value – Write PXI_TRG's status value

_8164_enable_pxi_input – Enable PXI input channel

_8164_select_pxi_output – Select PXI output channel

@ Description

_8164_get_pxi_trigger_value():

Reads back the PXI TRG's status value from bit0 to bit7.

_8164_set_pxi_trigger_value():

Writes the PXI TRG's status value from bit0 to bit7.

_8164_enable_pxi_input():

There are three dedicated channels in the PXI_TRG for STA, STP, and CEMG. When this is enabled, the corresponding channels of PXI_TRG are set as inputs. These channels map to PXI_TRG's channel 5, 6, and 7. If you need these functions, the corresponding PXI_TRG channel is fixed as Input.

_8164_select_pxi_output():

You can select many sources for PXI_TRG outputs. You can use this function to assign a source for PXI_TRG output. There should only be one source at the same time. The source could be from CMP1, CMP2, or other general purpose output pins from motion chip. The output channel of PXI_TRG could also be assigned using this function. The STAR_TRG could be assigned to an output channel in this function.

@ Syntax

C/C++ (DOS, Win32)

```
I16 _8164_get_pxi_trigger_value(I16 CardNo, U16
     *value);
I16 _8164_set_pxi_trigger_value(I16 CardNo, U16
     value);
I16 _8164_enable_pxi_input(I16 CardNo, I16 STA,
     I16 STP, I16 CEMG);
I16 _8164_select_pxi_output(I16 CardNo, I16
     source, I16 pxi_channel);
```

Visual Basic (Win32)

```
B_8164_get_pxi_trigger_value(ByVal CardNo As
     Integer, value As Integer) As Integer
B_8164_set_pxi_trigger_value(ByVal CardNo As
     Integer, ByVal value As Integer) As Integer
B_8164_enable_pxi_input(ByVal CardNo As Integer,
     ByVal STA As Integer, ByVal STP As Integer,
     ByVal CEMG As Integer) As Integer
B_8164_select_pxi_output(ByVal CardNo As Integer,
     ByVal source As Integer, ByVal pxi_channel
     As Integer) As Integer
```

@ Argument

CardNo: Designated card number

***value**: PXI_TRG value bit0-7, PXI_STAR value in bit8

value: PXI_TRG value bit0-7

STA: 0=disable, 1=link to PXI_TRG[5], 2=link to PXI_STAR

STP: 0=disable, 1=link to PXI_TRG[6]

CEMG: 0=disable, 1=link to PXI_TRG[7]

source:

- 0 = Software control
- 1 = AP6X
- 2 = AP6Y
- 3 = AP6Z
- 4 = AP6U
- 5 = CMP1
- 6 = CMP2

pxi_channel:

- PXI_TRG value bit0-7
- STAR_TRG bit8

@ Return Value

```
ERR_NoError
ERR_CardTypeWrong
ERR_CardNoError
ERR_PXISourceWrong
ERR_PXIChannelWrong
```

7 Connection Example

This chapter illustrates connection examples between the PCI-/MPC-/PXI-8164 card and servo drivers or stepping drivers.

7.1 General Wiring Description

CN1: Receives +24V power from the external power supply. (PCI-8164 only)

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

CN3: Receives pulse commands from manual pulse in the PCI-8164 card.

General Purpose DIO for MPC-8164

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

CN5: TTL digital output for PCI-8164This example will illustrate the connection between a Panasonic Servo Driver and the 8164. Figure 9 shows the wiring diagram.

NOTES:

1. For convenience, the drawing shows connections for single axis only.
2. Default pulse output mode is OUT/DIR. Default input mode is 1X AB phase. Other modes can be set using the available software functions.
3. Most general purpose servomotor drivers can operate in Torque Mode, Velocity Mode, or Position Mode. To connect the card, you must set the operating mode to Position Mode.

Figure 7-1 illustrates the integration of the PCI-8164 card with a physical system.

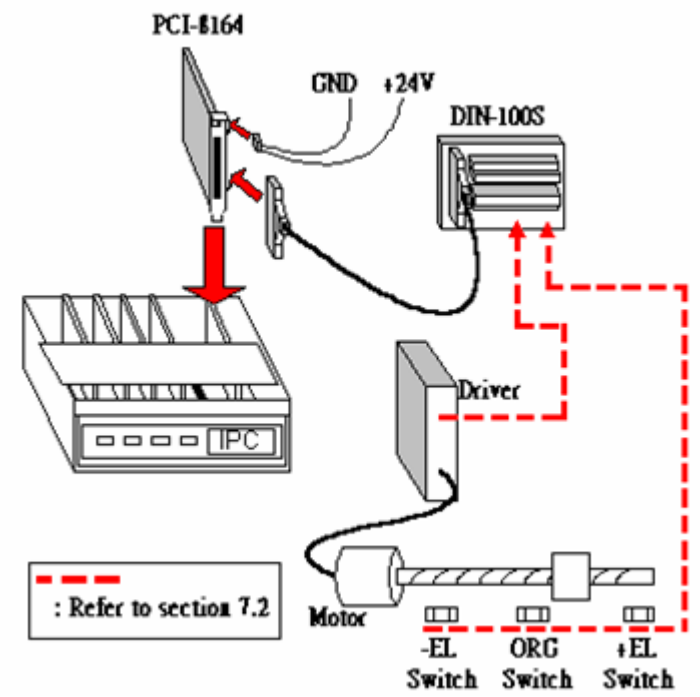


Figure 7-1: System Integration with PCI-8164

7.2 Connection Example with Servo Driver

This example illustrates the connection between a ***Panasonic Servo Driver*** and the card. Figure 7-2 shows the wiring diagram.

NOTES:

1. For convenience, the drawing shows connections for single axis only.
2. Default pulse output mode is OUT/DIR. Default input mode is 1X AB phase. Other modes can be set using the available software functions.
3. Most general purpose servomotor drivers can operate in Torque Mode, Velocity Mode, or Position Mode. To connect the card, you must set the operating mode to Position Mode.

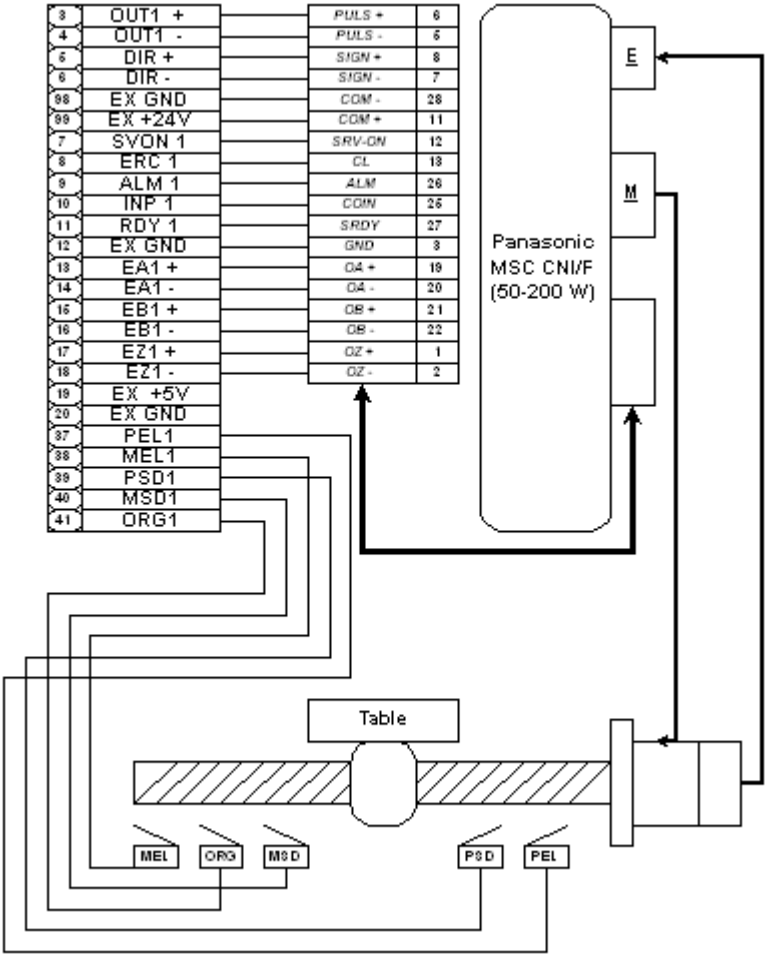


Figure 7-2: Connection of PCI-8164 with Panasonic Driver

Wiring of PCI-8164 with SANYO AC Servo PY2

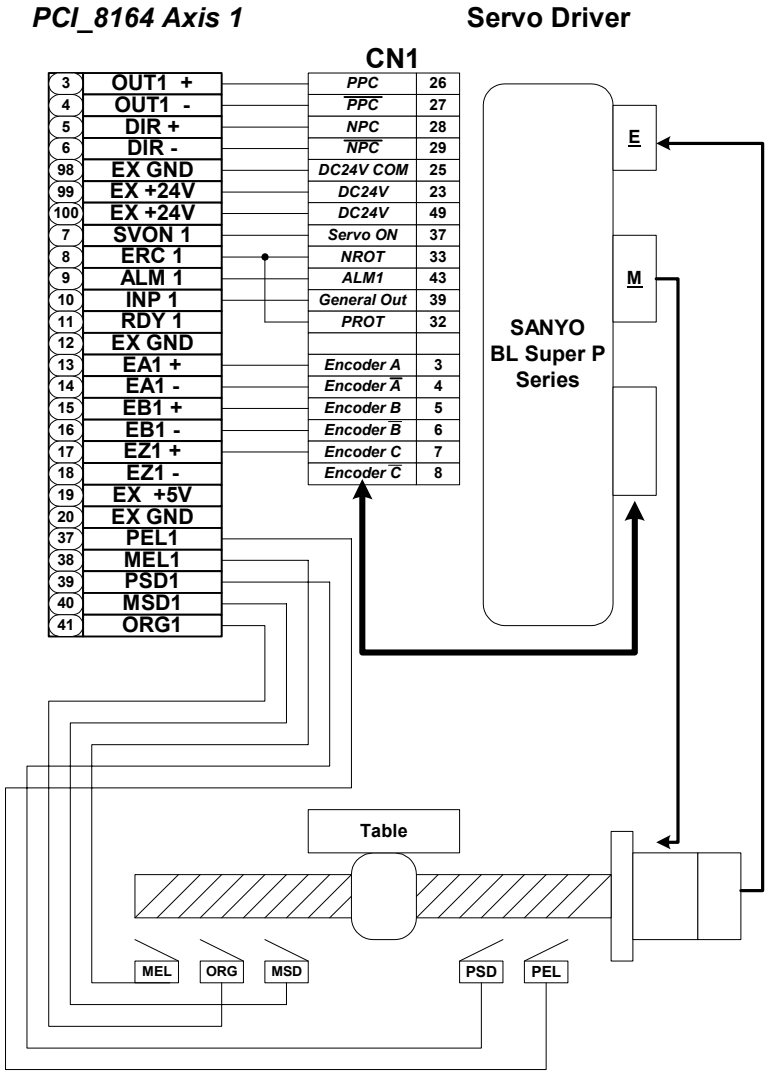


Figure 7-3: Connection of PCI-8164 with SANYO Driver

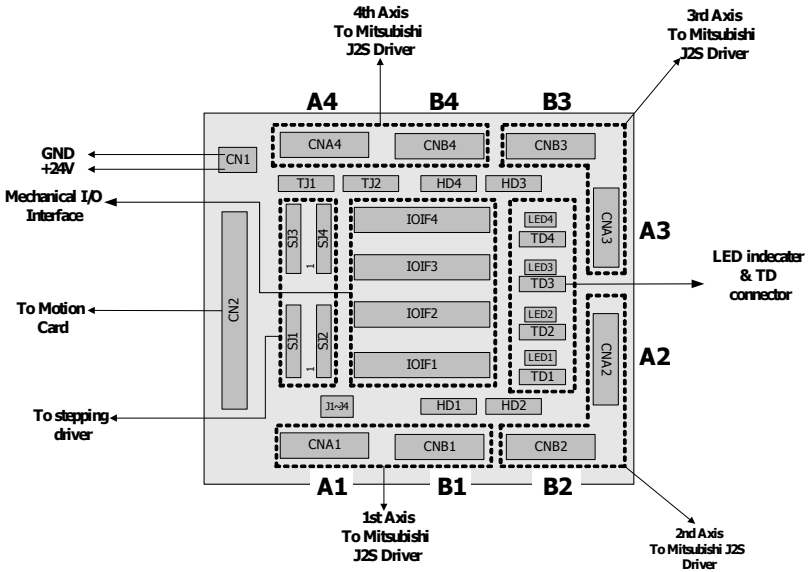
7.3 Wiring with DIN-814M

WARNING

The DIN-814M is used for wiring between Mitsubishi J2S series servo drivers / stepper with pulse trains input driver and ADLINK PCI-8134, PCI-8164, PXI-8164 or MPC-8164 motion controller card ONLY. Never use it with any other servo driver or cards.

NOTES

1. **Servo and stepper**: The DIN-814M provides two connection methods for each axis: CNA and CNB connectors for Mitsubishi J2S series servo drivers, and an SJ connector for stepping drivers. The signals in SJ, CNA, and CNB of the same axis are directly shorted. DO NOT use both connectors at the same time.



2. **CAN or CNB cables**: Two 20-pin cables (pin-to-pin) are required to connect the CNA and CNB with the Mitsubishi J2S driver. The required cables are available from ADLINK.
3. **PSD Signal**: The PSD signal on the IOIF# connector has different functions depending on different motion

cards. When using PCI-8134, PSD is used as a positive slow down signal. When using PCI-8164/PXI-8164/MPC-8164, PSD1/2 is for CMP1/2 of Axis0/Axis1 and PSD3/4 is for LTC3/4 of Axis3/Axis4.

4. **Emergency Signal**: This signal is defined by Mitsubishi's servo driver (pin15 of CNB) as a normally closed input. Disable the signal by connecting jumpers J1-J4 of 1-2 to GND. You may also connect a switch to the EMG pin of each axis of the IOIF# or HD# connectors and short jumpers J1-J4 of 2-3 to make EMG controllable by a switch.

7.3.1 PIN Assignments:

CNA1-CNA4 (Mitsubishi AC Servo Driver CNA Interface)

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

CNB1-CNB4 (Mitsubishi AC Servo Driver CNB Interface)

No.	Name	I/O	Function	No.	Name	I/O	Function
1	IGND	--	Isolated Ground	2	VC	I	Analog Torque Command
3	+24V	--	Driver Voltage output	4	DO1	O	ABS bit 0
5	Servo ON	I	Servo On	6	TLC	I	Limiting Torque
7	SP2	I	Speed Selection 2	8	ABSM	I	ABS transfer mode
9	ABSR	I	ABS request	10	IGND	--	Isolated Ground
11	P15R	--	+15 VDC power supply	12	VLA	I	Analog Speed Limit
13	+24V	O	Driver Voltage output	14	RES	I	Reset
15	EMG	--	Internal EMG Signal	16	IGND	--	Isolated Ground
17	IGND	--	Isolated Ground	18	ALM	I	Servo Alarm
19	ZSP	I	Zero Speed	20	IGND	--	Isolated Ground

IOIF1-IOIF4 External Motion Input Signal Interface)

No.	Name	I/O	Function	No.	Name	I/O	Function
1	+24V	O	Voltage output	6	MSD	I	Negative Slow Switch (+)
2	EX_EMG	I	External EMG Signal	7	ORG	I	Origin Switch
3	PEL	I	Positive Limit (+)	8	IGND	--	Isolated Ground
4	MEL	I	Negative Limit (-)	9	IGND	--	Isolated Ground
5	PSD	I	*CMP/LTC/PSD		* Please check the note in first page		

SJ1-SJ4 (Stepping Motor Control Interface)

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

CN1 (External +24VDC Input Connector)

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

HD1-HD4 (Auxiliary. Servo I/O Interface)

Name	I/O	Function	No.	Name	I/O
+24V	O	Voltage output	4	EX_EMG	I
Servo ON	O	Servo On	5	ALM	I
RDY	I	Servo Ready	6	IGND	--

Jumper (Mitsubishi AC Servo Driver EMG Signal Source Selection)

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

TD1-TD4 (External AC Servo Driver Reset & ABS system Interface)

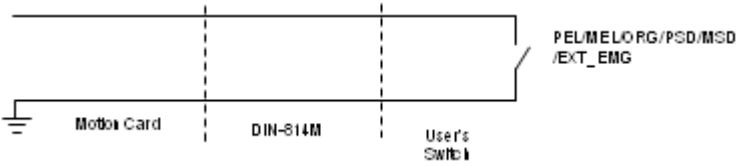
No.	Name	I/O	Function	No.	Name	I/O	Function
1	RES	I	Reset	5	ZSP	O	ABS bit 1
2	ABSM	I	ABS transfer mode	6	TLC	O	Send data ready
3	ABSR	I	ABS request	7	IGND	--	Isolated Ground
4	D01	O	ABS bit 0	8	IGND	--	Isolated Ground

TJ1-TJ2 (Speed Control Mode Interface only for Axis0,1)

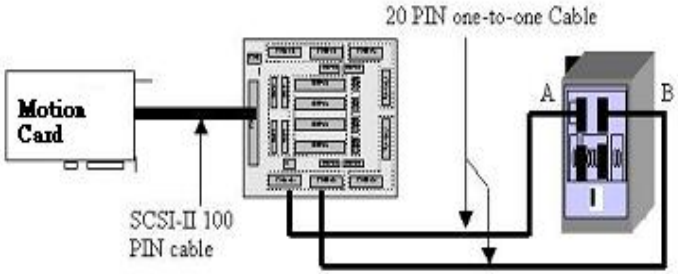
No.	Name	I/O	Function	No.	Name	I/O	Function
1	SP2	I	Speed Selection 2	4	LG	--	Control Common
2	P15R	I	+15VDC power supply	5	TLA	I	Analog torque limit
3	VC	O	Analog Speed Command	6	LG	--	Control Common

7.3.2 Signal Connections

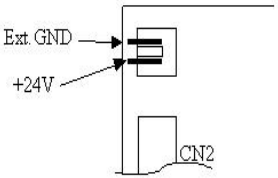
1. PEL, MEL, ORG, PSD, MSD (in IOIF#)
2. EX_EMG (both IOIF# and HD#)



3. CMP, LTC (in IOIF#'s PSD pin only for 8164 series)

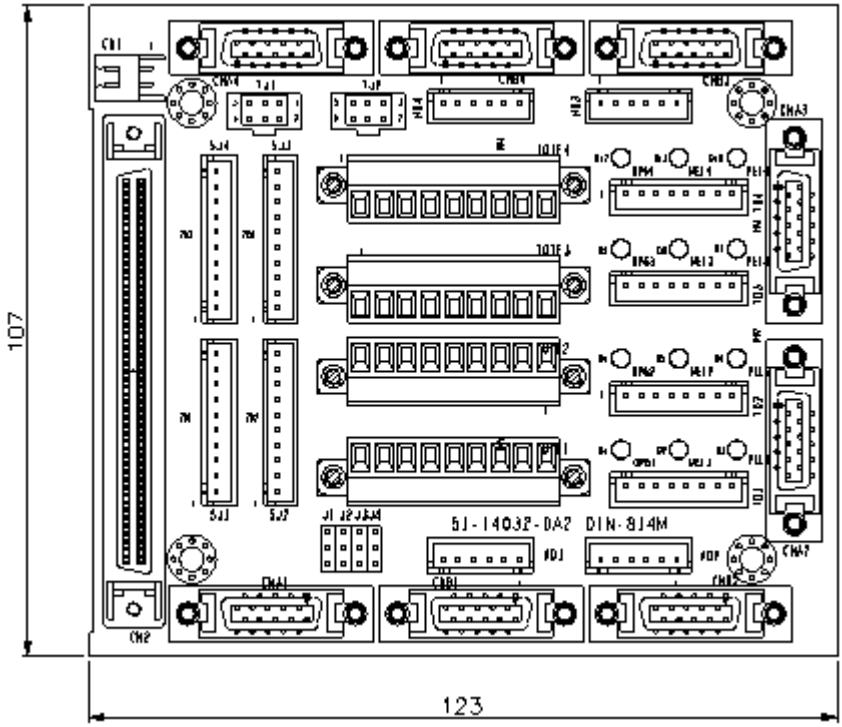


4. CNA & CNB, CN2
5. CN1: This connector is parallel with on board CN1.



6. SJ1-4: For stepping driver, refer to the ADLINK user's manual for wiring.
7. TD1-4: Refer to the Mitsubishi user's manual for wiring.
8. TJ1-2: Refer to the Mitsubishi user's manual for wiring.

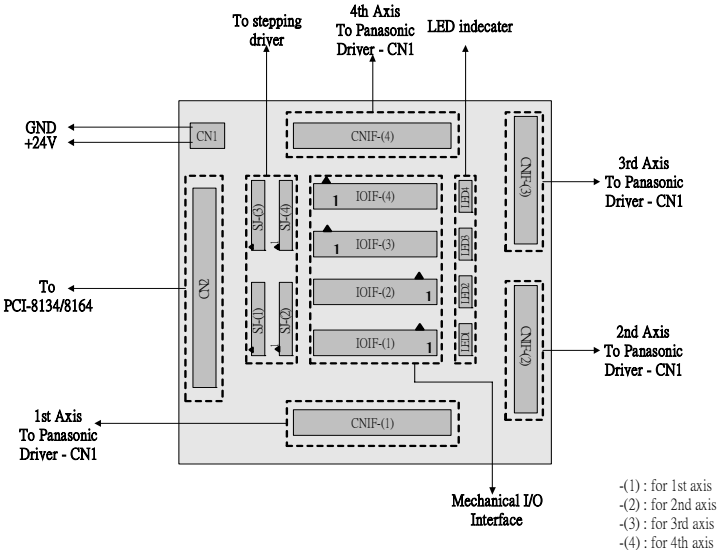
7.3.3 Mechanical Dimensions:



7.4 Wiring with DIN-814P

WARNING

The DIN-814M is used for wiring between the **Panasonic MINAS MSD** series servo driver and ADLINK **PCI-8134** or **PCI-8164** motion controller cards ONLY. Do not try the DIN-814P to connect other servo drivers or cards.

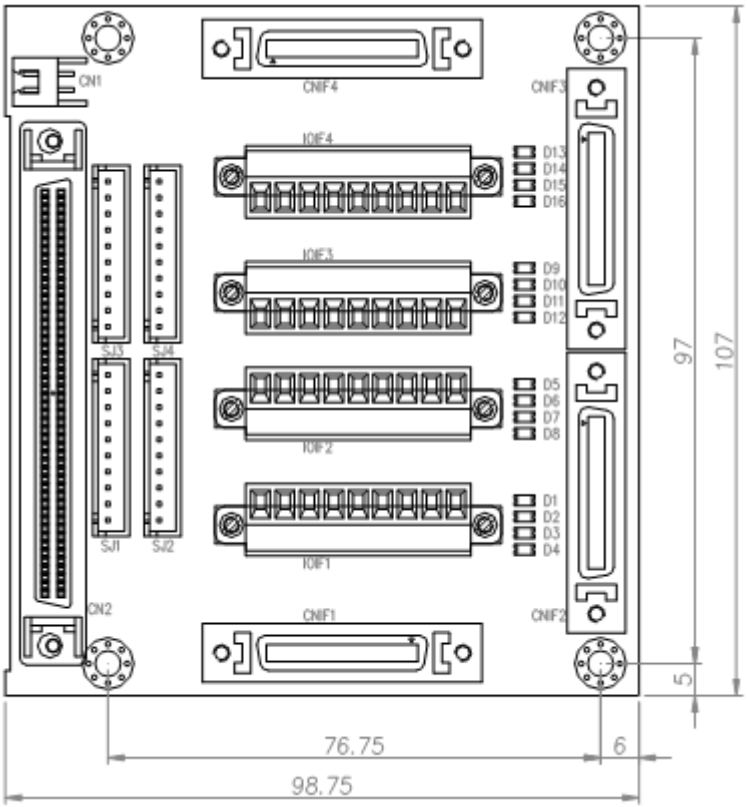


NOTES

1. The DIN-814P provides TWO connection methods for every axis. The first is through the CNIF connector for the Panasonic MINAS MSD series servo driver. The second is through the SJ connector for stepping drivers or other servo drivers (for the Mitsubishi J2S driver, use DIN-814M). The signals in SJ and CNIF of the same axis are directly shorted. DO NOT use these connectors at the same time.

2. A 36-pin cable (one-to-one) is required to connect the CNIF and the Panasonic MINAS MSD driver. Contact your local ADLINK representative for availability.
3. Depending on the PCI-8134/PCI-8164 card usage, some signals in the IOIF connector, such as PSD and MSD, will function differently. When using PCI-8134, the PSD and MSD signals are for positive slow down and negative slow down signal, respectively. When using PCI-8164, PSD is for CMP and LTC, and MSD is for SD. For more details, refer to the PCI-8134/PCI-8164 user's manual.

7.4.1 Mechanical Dimensions:



7.4.2 PIN Assignment:

CNIF1-CNIF4

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

IOIF1-IOIF4

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

SJ1-SJ4

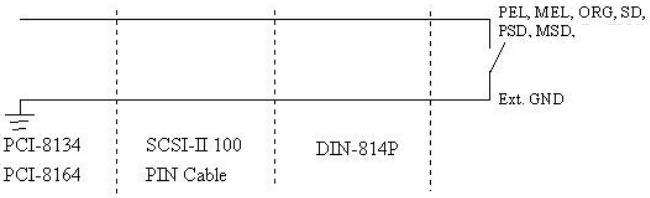
No.	Name	I/O	Function	No	Name	I/O	Function
1	OUT+	O	Pulse Signal (+)	6	ALM	I	Servo Alarm
2	OUT-	O	Pulse Signal (-)	7	+5V	O	Voltage output
3	DIR+	O	Direction Signal (+)	8	Servo ON	O	Servo On
4	DIR-	O	Direction Signal (-)	9	+5V	O	Voltage output
5	EZ+	I	Index Signal	10	IGND	--	Isolated Ground

CN1

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

7.4.3 How to wire

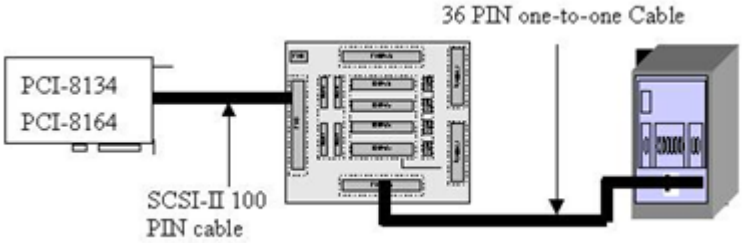
PEL, MEL, ORG, SD, PSD, MSD (in IOIF):



CMP, LTC (in IOIF)

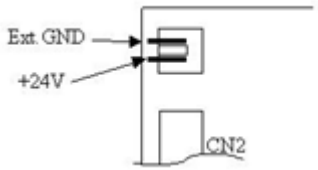
- CMP is a TTL 5V or 0V output (vs. Ext GND)
- LTC is a TTL 5V or 0V input (vs. Ext. GND)

CNA & CNB, CN2



SJ: Refer to PCI-8134/PCI-8164 user's manual for wiring information.

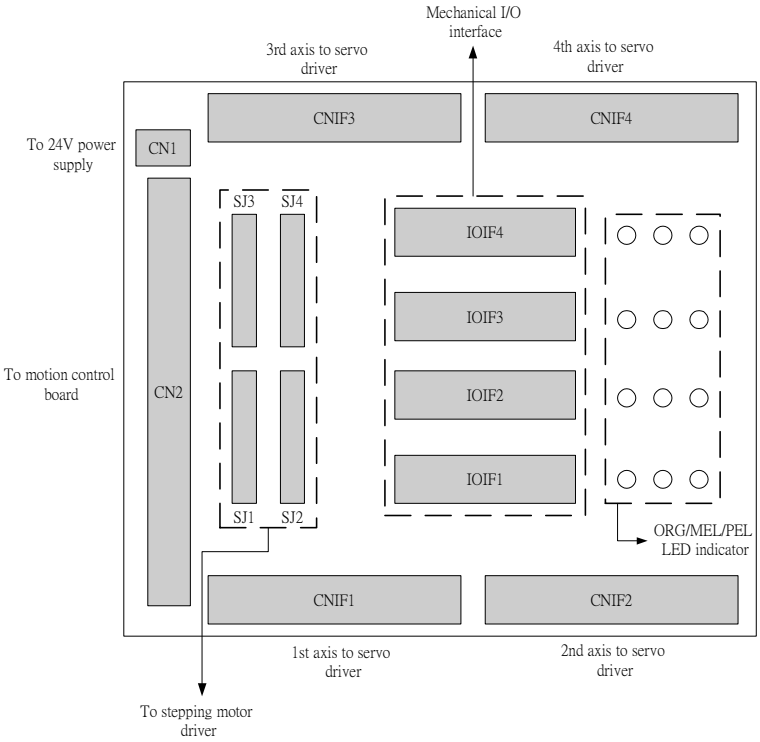
CN1:



7.5 Wiring with DIN-814PA

WARNING

The DIN-814PA is designed for Panasonic MINAS A-series servo drivers with ADLINK PCI-8134/PCI-8164 series motion control board. DO NOT use the DIN-814PA with other servo drivers and motion control boards.



Description:

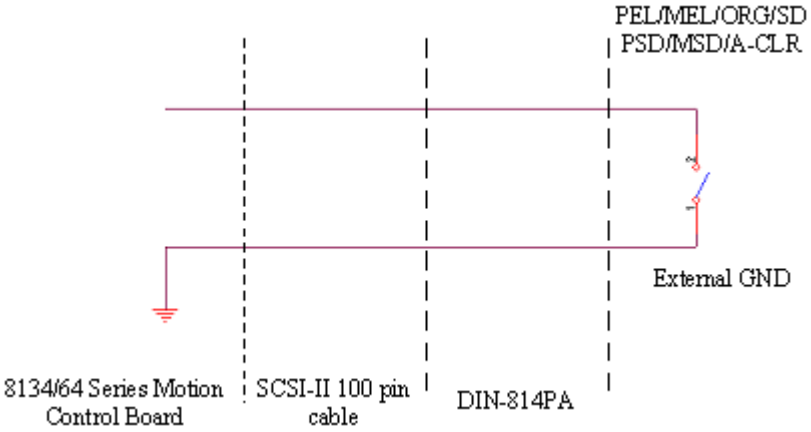
1. The DIN-814PA has two kinds of connections for four axes motion control. One is connector CNIF# designed for Panasonic MINAS A-series servo driver. Another is the SJ# connector designed for stepping drivers or other

servo drivers. The signals at CNIF# and SJ# are connected together. DO NOT use them simultaneously.

2. A 50-pin cable is required to connect the Panasonic MINAS A-series servo driver with the CNIF# connector on the DIN-814PA. Contact your local ADLINK representative for additional information.
3. Signals PSD# and MSD# at the connector IOIF# are assigned different names on the 8164-series motion control board. The PSD1, PSD2, PSD3, and PSD4 signals are renamed CMP1, CMP2, LTC3, and LTC4, respectively. MSD# is replaced by SD/PCS#.

7.5.1 Wiring Example:

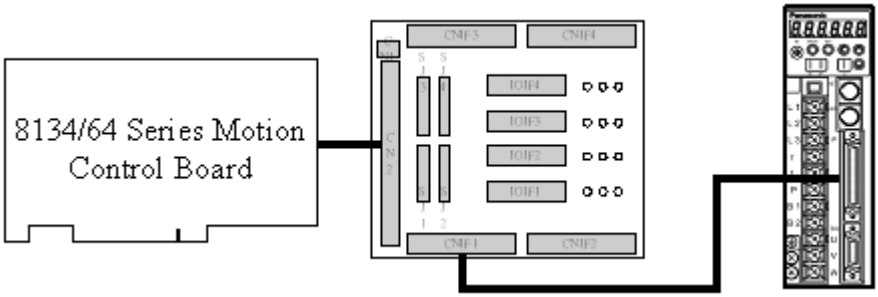
PEL, MEL, ORG, SD, PSD, MSD, and Alarm Clear (at IOIF):



CMP, LTC (at IOIF)

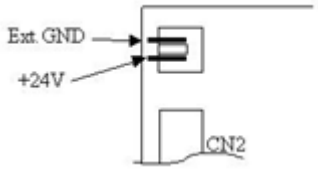
- ▷ CMP is a TTL 5V or 0V output (vs. Ext GND)
- ▷ LTC is a TTL 5V or 0V input (vs. Ext. GND)

CNIF# and CN2

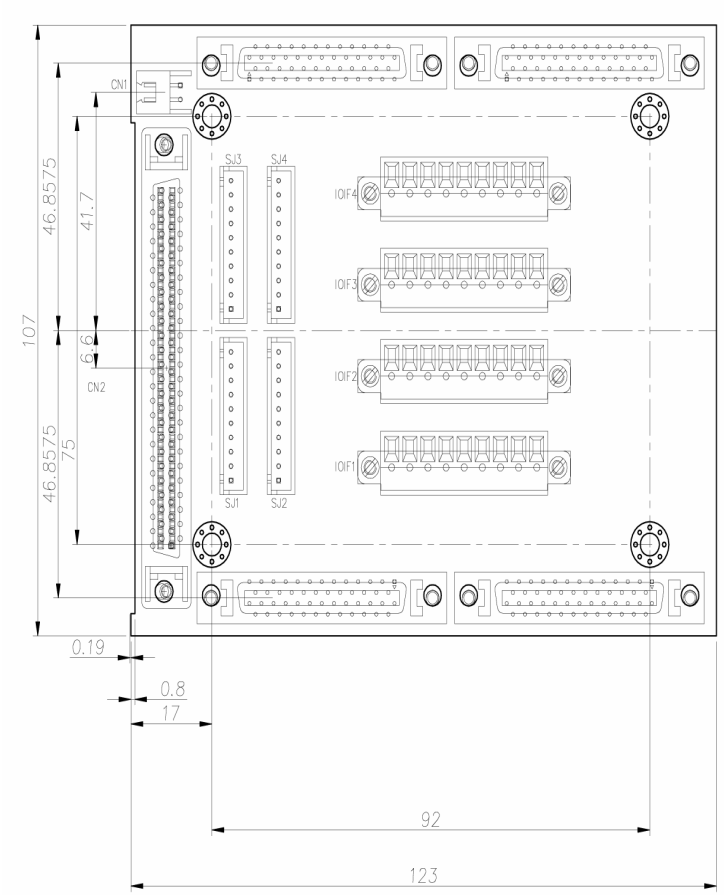


SJ: Refer to 8134/64 series user manual.

CN1: 24V power supply input



7.5.2 Mechanical Dimensions:



7.5.3 PIN Assignment:

CNIF1-CNIF4

No.	Name	I/O	Function	No.	Name	I/O	Function
1	NC	--		2	NC	--	
3	OUT-	O	Pulse Signal (-)	4	OUT+	O	Pulse Signal (+)
5	DIR-	O	Direction Signal (-)	6	DIR+	O	Direction Signal (+)
7	+24V	O	+24V Power supply	8	NC	--	
9	NC	--		10	NC	--	
11	NC	--		12	NC	--	
13	IGND	--	Isolated Ground	14	NC	--	
15	IGND	--	Isolated Ground	16	NC	--	
17	IGND	--	Isolated Ground	18	NC	--	
19	NC	--		20	NC	--	
21	EA+	I	Encoder A-phase (+)	22	EA-	I	Encoder A-phase (-)
23	EZ+	I	Encoder Z-phase (+)	24	EZ-	I	Encoder Z-phase (-)
25	IGND	--	Isolated Ground	26	NC	--	
27	NC	--		28	NC	--	
29	SVON		Servo On	30	ERC	O	
31	A-CLR	O	Alarm Clear	32	NC	--	
33	IGND	--	Isolated Ground	34	IGND	--	Isolated Ground
35	RDY		Servo Ready	36	IGND	--	Isolated Ground
37	ALM		Servo Alarm	38	IGND	--	Isolated Ground
39	INP		Servo In Position	40	NC	--	
41	IGND	--	Isolated Ground	42	NC	--	
43	NC	--		44	NC	--	
45	NC	--		46	NC	--	
47	NC	--		48	EB+	I	Encoder B-phase (+)
49	EB-	I	Encoder B-phase (-)	50	NC	--	

IOIF1-IOIF4

No.	Name	I/O	Function	No.	Name	I/O	Function
1	+24V	O	+24V Power supply	6	MSD	I	Negative Slow Switch (+)
2	+24V	O	External EMG Signal	7	ORG	I	
3	PEL	I	Positive Limit (+)	8	A-CLR	O	Alarm Clear
4	MEL	I	Negative Limit (-)	9	IGND	--	Isolated Ground
5	PSD	I	Positive Slow Switch (+)				

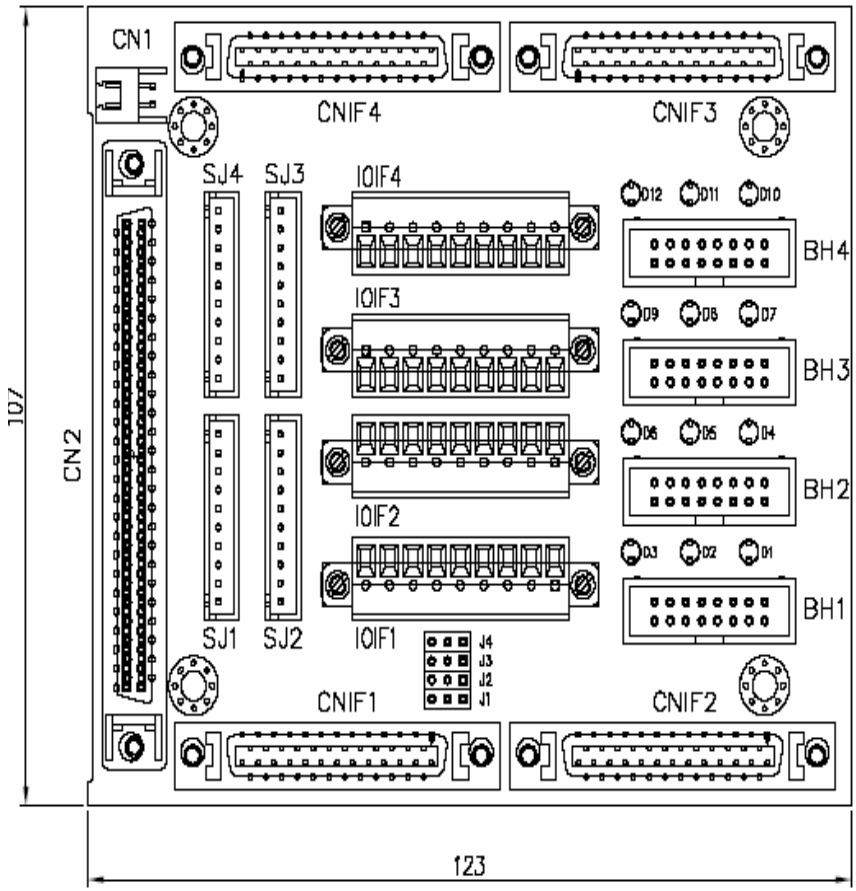
SJ1-SJ4

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

CN1

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

7.6 Wiring with DIN-814M-J3A



7.6.1 PIN Assignment:

CNIF1-CNIF4

No.	Name	I/O	Function	No.	Name	I/O	Function
1	P15R	--	15VDC power supply	2	VLA	O	Analog speed limit
3	IGND	--	Isolated Ground	4	EA+	I	Encoder A-phase(+)
5	EA-	I	Encoder A-phase(-)	6	EB+	I	Encoder B-phase(+)
7	EB-	I	Encoder B-phase(-)	8	EZ+	I	Encoder Z-phase(+)
9	EZ-	I	Encoder Z-phase(-)	10	OUT+	O	Pulse Signal(+)
11	OUT-	O	Pulse Signal(-)	12	N.C		
13	N.C			14	N.C		
15	SVON	O	Servo ON	16	SP2	O	Speed selection 2
17	ABSM	O	Forward rotation	18	ABSR	O	Reverse rotation
19	RES	I	Reset Signal	20	+24V	--	Voltage output
21	+24V	--	Voltage output	22	ABSB0	I	Speed reached
23	ZSP	I	Zero Speed	24	INP	I	In-position Signal
25	TLC	I	Limiting Torque	26	N.C		
27	TC	O	Analog torque command	28	IGND	--	Isolated Ground
29	N.C			30	N.C		
31	N.C			32	N.C		
33	N.C			34	IGND	--	Isolated Ground
35	DIR+	O	Direction Signal(+)	36	DIR-	O	Direction Signal(-)
37	N.C			38	N.C		
39	N.C			40	N.C		
41	ERC	O	Error counter Clear	42	EMG	O	Emergency stop
43	IGND	--	Isolated Ground	44	IGND	--	Isolated Ground
45	LOP	O	Control selection	46	IGND	--	Isolated Ground
47	IGND	--	Isolated Ground	48	ALM	I	Servo Alarm Signal
49	RDY	I	Servo Ready Signal	50	N.C		

BH1-BH4

No.	Name	I/O	Function	No.	Name	I/O	Function
1	RES	I	Reset Signal	2	ABSM	O	Forward rotation
3	ABSR	O	Reverse rotation	4	ABSB0	I	Speed reached
5	ZSP	I	Zero Speed	6	TLC	I	Limiting Torque
7	SP2	O	Speed selection 2	8	P15R	--	15VDC power supply
9	VLA	O	Analog speed limit	10	TC	O	Analog torque command
11	+24V	--	Voltage output	12	ALM	I	Servo Alarm Signal
13	EXEMG	I	External EMG Signal	14	IGND	--	Isolated Ground
15	LOP	O	Control selection	16	IGND	--	Isolated Ground

IOIF1-IOIF4

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

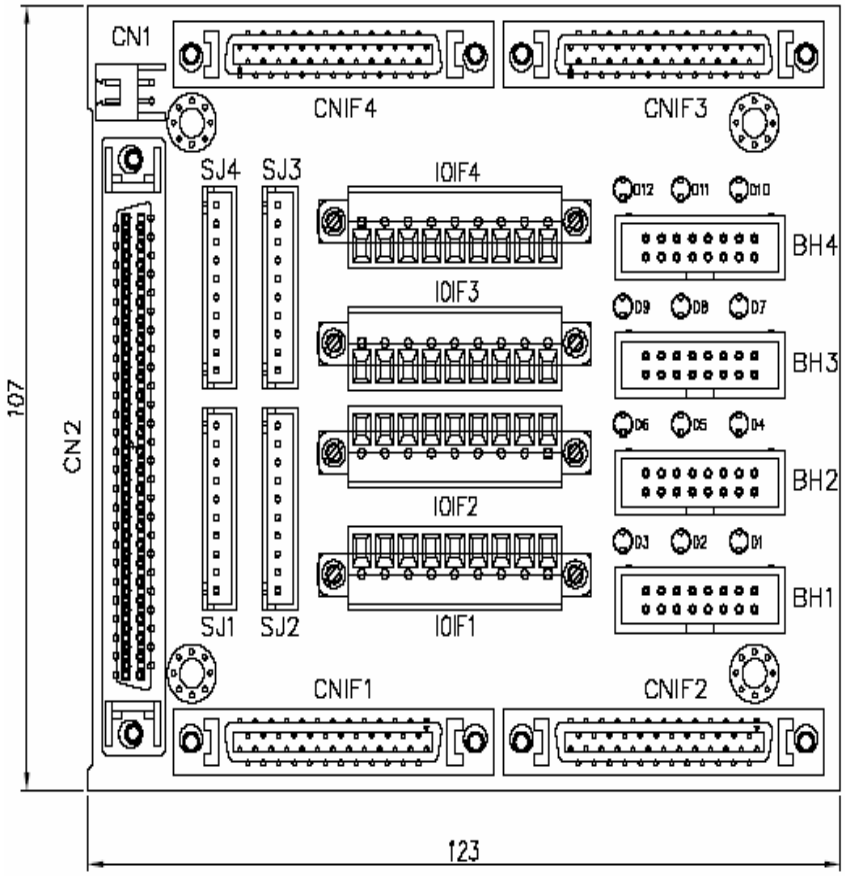
SJ1-SJ4

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

CN1

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

7.7 Wiring with DIN-814Y



7.7.1 PIN Assignment:

CNIF1-CNIF4

No.	Name	I/O	Function	No.	Name	I/O	Function
1	IGND	--	Isolated Ground	2	IGND	--	Isolated Ground
3	PL1	I	Collector +12V Output	4	SEN	I	Reset Absolute Encoder
5	V-REF	I	Speed Command	6	IGND	--	Isolated Ground
7	OUT+	O	Pulse Signal (+)	8	OUT-	O	Pulse Signal (-)
9	T-REF	I	Torque Command	10	IGND	--	Isolated Ground
11	DIR+	O	Direction Signal (+)	12	DIR-	O	Direction Signal (-)
13	X			14	IGND	--	Isolated Ground
15	ERC	O	Dev. ctr, clr. Signal	16	X		
17	X			18	X		
19	EZ+	I	Encoder Z-phase (+)	20	EZ-	I	Encoder Z-phase (-)
21	BAT+	I	Battery (+)	22	BAT-	I	Battery (-)
23	X			24	X		
25	INP	I	In-position Signal	26	IGND	--	Isolated Ground
27	X			28	X		
29	RDY	I	Servo Ready Signal	30	IGND	--	Isolated Ground
31	ALM	I	Alarm Signal	32	IGND	--	Isolated Ground
33	EA+	I	Encoder A-phase (+)	34	EA-	I	Encoder A-phase (-)
35	EB+	I	Encoder B-phase (+)	36	EB-	I	Encoder B-phase (-)
37	AL01	O	Alarm Number1	38	AL02	O	Alarm Number2
39	AL03	O	Alarm Number3	40	SVON	O	Servo ON
41	P-CON	I	Proportion control	42	IGND	--	Isolated Ground
43	IGND	--	Isolated Ground	44	RES	I	Reset Signal
45	/P-CL	I	Forward Current	46	/N-CL	I	Reverse Current
47	+24V	O	Voltage output	48	PSO+	O	Encoder S-phase (+)
49	PSO-	O	Encoder S-phase (-)	50	X		

BH1-BH4

No.	Name	I/O	Function	No.	Name	I/O	Function
1	SEN	I	Reset Absolute Encoder	2	BAT+	I	Battery (+)
3	BAT-	I	Battery (-)	4	PSO+	O	Encoder S-phase (+)
5	PSO-	O	Encoder S-phase (-)	6	P-CON	I	Proportion control
7	V-REF	I	Speed Command	8	T-REF	I	Torque Command
9	/P-CL	I	Forward Current	10	/N-CL	I	Reverse Current
11	AL01	O	Alarm Number1	12	AL02	O	Alarm Number2
13	AL03	O	Alarm Number3	14	PL1	I	Collector +12V Output
15	+24V	O	Voltage output	16	IGND	--	Isolated Ground

IOIF1-IOIF4

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

SJ1-SJ4

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

CN1

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

8 Appendix

8.1 Color code of CN3 Cable (MPC-8164 Only)

CN3 Pin No	Signal Name	Color	CN3 Pin No	Signal Name	Color
1	DOCOM	Brown	2	DOCOM	Pink-Black
3	DOCOM	Grey	4	DOCOM	Blue-White
5	DO0	Red	6	DO1	Grey-Black
7	DO2	White	8	DO3	Purple-White
9	DO4	Orange	10	DO5	Light Green-Black
11	DO6	Pink	12	DO7	White-Blue
13	--	Yellow	14	DICOM	Light Blue-Black
15	DICOM	Light Blue	16	DICOM	Red-White
17	DICOM	Green	18	DI0	Green-Black
19	DI1	Light Green	20	DI2	Brown-White
21	DI3	Blue	22	DI4	Yellow-Black
23	DI5	Red Black	24	DI6	White-Black
25	DI7	Purple	26	--	Black-Orange

Warranty Policy

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

1. Before using ADLINK's products please read the user manual and follow the instructions exactly. When sending in damaged products for repair, please attach an RMA application form which can be downloaded from: http://rma.adlinktech.com/policy/.
2. All ADLINK products come with a limited two-year warranty, one year for products bought in China:
   - The warranty period starts on the day the product is shipped from ADLINK's factory.
   - Peripherals and third-party products not manufactured by ADLINK will be covered by the original manufacturers' warranty.
   - For products containing storage devices (hard drives, flash cards, etc.), please back up your data before sending them for repair. ADLINK is not responsible for any loss of data.
   - Please ensure the use of properly licensed software with our systems. ADLINK does not condone the use of pirated software and will not service systems using such software. ADLINK will not be held legally responsible for products shipped with unlicensed software installed by the user.
   - For general repairs, please do not include peripheral accessories. If peripherals need to be included, be certain to specify which items you sent on the RMA Request & Confirmation Form. ADLINK is not responsible for items not listed on the RMA Request & Confirmation Form.

3. Our repair service is not covered by ADLINK's guarantee in the following situations:
   - Damage caused by not following instructions in the User's Manual.
   - Damage caused by carelessness on the user's part during product transportation.
   - Damage caused by fire, earthquakes, floods, lightening, pollution, other acts of God, and/or incorrect usage of voltage transformers.
   - Damage caused by unsuitable storage environments (i.e. high temperatures, high humidity, or volatile chemicals).
   - Damage caused by leakage of battery fluid during or after change of batteries by customer/user.
   - Damage from improper repair by unauthorized ADLINK technicians.
   - Products with altered and/or damaged serial numbers are not entitled to our service.
   - This warranty is not transferable or extendible.
   - Other categories not protected under our warranty.
4. Customers are responsible for shipping costs to transport damaged products to our company or sales office.
5. To ensure the speed and quality of product repair, please download an RMA application form from our company website: http://rma.adlinktech.com/policy. Damaged products with attached RMA forms receive priority.

If you have any further questions, please email our FAE staff: service@adlinktech.com.

OCEAN CHIPS
Океан Электроники
Поставка электронных компонентов

Компания «Океан Электроники» предлагает заключение долгосрочных отношений при поставках импортных электронных компонентов на взаимовыгодных условиях!

Наши преимущества:

- Поставка оригинальных импортных электронных компонентов напрямую с производств Америки, Европы и Азии, а так же с крупнейших складов мира;
- Широкая линейка поставок активных и пассивных импортных электронных компонентов (более 30 млн. наименований);
- Поставка сложных, дефицитных, либо снятых с производства позиций;
- Оперативные сроки поставки под заказ (от 5 рабочих дней);
- Экспресс доставка в любую точку России;
- Помощь Конструкторского Отдела и консультации квалифицированных инженеров;
- Техническая поддержка проекта, помощь в подборе аналогов, поставка прототипов;
- Поставка электронных компонентов под контролем ВП;
- Система менеджмента качества сертифицирована по Международному стандарту ISO 9001;
- При необходимости вся продукция военного и аэрокосмического назначения проходит испытания и сертификацию в лаборатории (по согласованию с заказчиком);
- Поставка специализированных компонентов военного и аэрокосмического уровня качества (Xilinx, Altera, Analog Devices, Intersil, Interpoint, Microsemi, Actel, Aeroflex, Peregrine, VPT, Syfer, Eurofarad, Texas Instruments, MS Kennedy, Miteq, Cobham, E2V, MA-COM, Hittite, Mini-Circuits, General Dynamics и др.);

Компания «Океан Электроники» является официальным дистрибьютором и эксклюзивным представителем в России одного из крупнейших производителей разъемов военного и аэрокосмического назначения «**JONHON**», а так же официальным дистрибьютором и эксклюзивным представителем в России производителя высокотехнологичных и надежных решений для передачи СВЧ сигналов «**FORSTAR**».

AVIC JONHON

«JONHON» (основан в 1970 г.)

Разъемы специального, военного и аэрокосмического назначения:

(Применяются в военной, авиационной, аэрокосмической, морской, железнодорожной, горно- и нефтедобывающей отраслях промышленности)

«FORSTAR» (основан в 1998 г.)

ВЧ соединители, коаксиальные кабели, кабельные сборки и микроволновые компоненты:

(Применяются в телекоммуникациях гражданского и специального назначения, в средствах связи, РЛС, а так же военной, авиационной и аэрокосмической отраслях промышленности).

FORSTAR®

Телефон: 8 (812) 309-75-97 (многоканальный)
Факс: 8 (812) 320-03-32
Электронная почта: ocean@oceanchips.ru
Web: http://oceanchips.ru/
Адрес: 198099, г. Санкт-Петербург, ул. Калинина, д. 2, корп. 4, лит. А