PCI-8144

4-Axis Stepper
Motion Control Card
User's Manual

Manual Rev. 2.01

Revision Date: September 2, 2008

Part No: 50-11144-1010

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

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

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

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

Table of Contents

List of Tables

List of Figures

1 Introduction

The PCI-8144 is a cost-effective 4-axis motion controller card with a PCI interface that can generate a pulse train up to 2.4MHz to control a stepper motor. As a motion controller, it provides non-symmetric acceleration and deceleration speed profile settings, T-curve and S-surve speed profile control, and simultaneously start/stop. Changing speed on the fly is also available with a single axis operation.

Multiple PCI-8144 cards can be installed in one system. The on-board switch can be used to the set a specific index for each board in order to manage multiple cards.

Figure 1 shows the functional block diagram of the PCI-8144 card. All functions and computations are performed internally by the ASIC, thus limiting the impact on the PC's CPU.

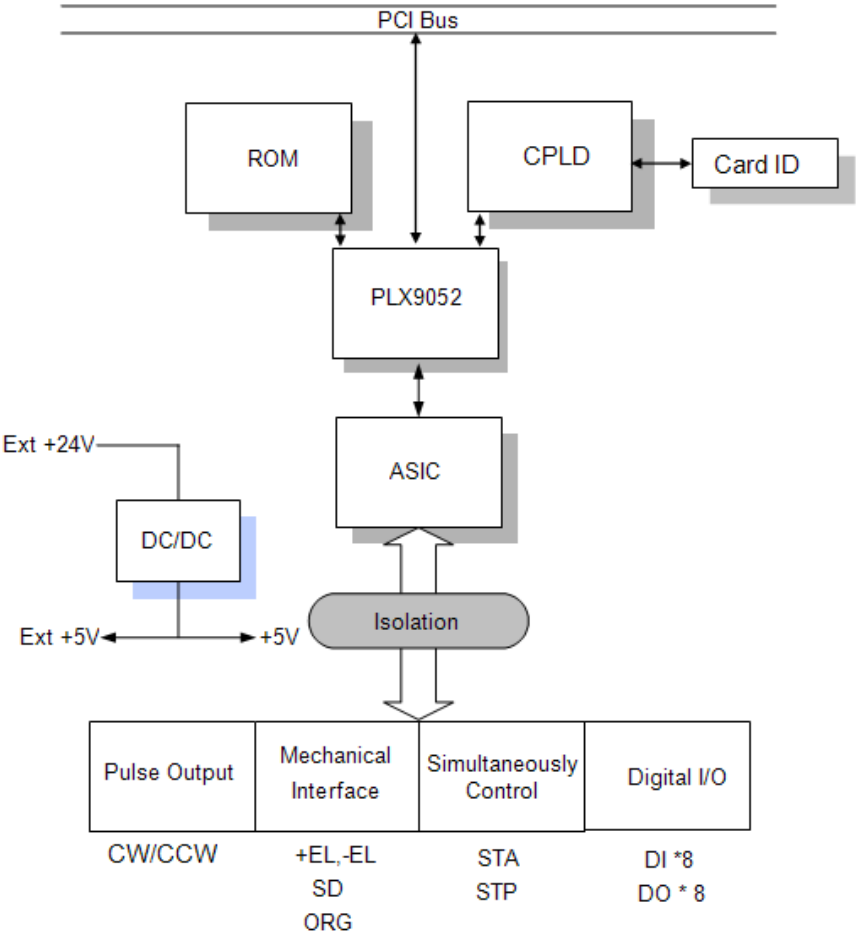


Figure 1-1: PCI-8144 Block Diagram

MotionCreatorPro is a Windows-based application development software package included with the PCI-8144 and is useful for debugging a motion control system during the design phase of a project. An on-screen display lists information of all installed axes and I/O signal status of the PCI-8144.

Windows programming libraries are also included for C++ and Visual Basic. Sample programs are provided to illustrate the operations of the functions.

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

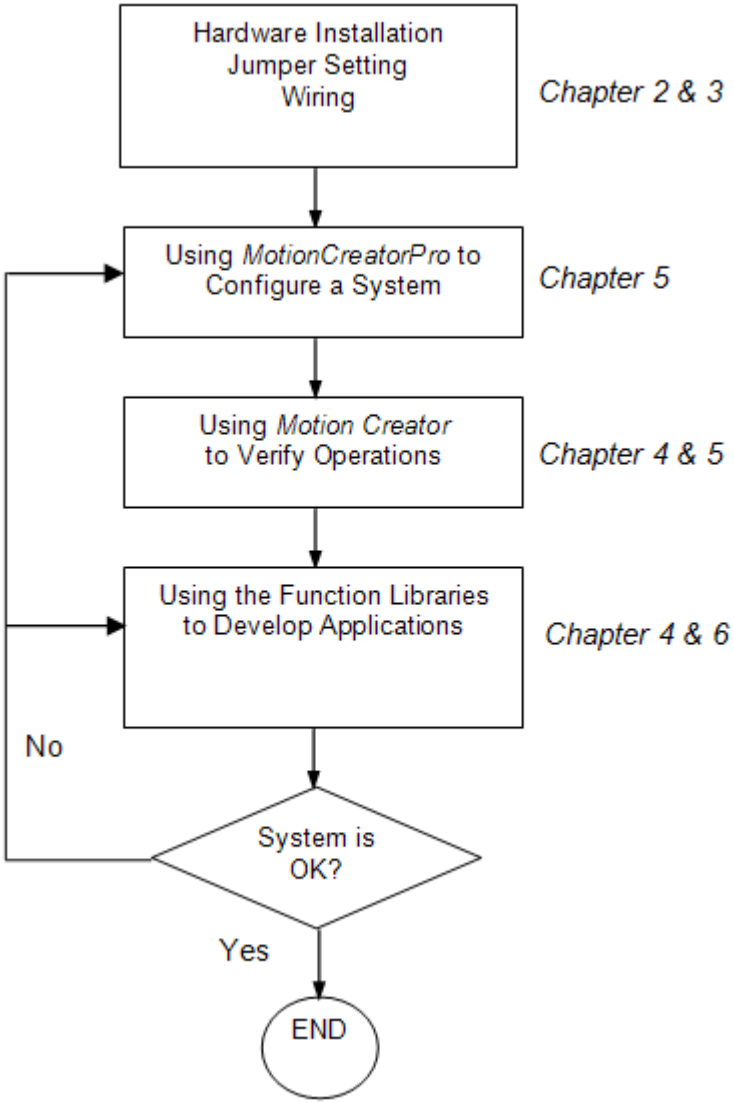


Figure 1-2: Flow chart for building an application

1.1 Features

The following list summarizes the main features of the PCI-8144 motion control system.

- 32-bit PCI bus Plug and Play
- 4 pulse train channel for stepping motors
- Maximum output frequency: up to 2.4MPPS
- Pulse output options: CW/CCW
- Programmable acceleration and deceleration time for all modes
- Trapezoidal and S-curve velocity profiles for all modes
- Change speed on the fly
- Home return modes with ORG & SD signal
- Hardware backlash compensator and vibration suppression
- Card index setting by switch
- All digital input and output signals are 2500Vrms isolated
- Programmable interrupt sources
- Simultaneous start/stop
- Software supports a maximum of up to 12 PCI-8144 cards operation in one system
- Includes *MotionCreatorPro*, a Microsoft Windows-based application development software
- PCI-8144 libraries and utilities for Windows 2000/XP.

1.2 Specifications

Applicable Motors

- ▷ Stepping motors
- ▷ Excitation sequencing output for 2-phase stepping motor with JP2 connector

Performance

- ▷ Number of controllable axes: 4
- ▷ Maximum pulse output frequency: 2.4MPPS, trapezoidal, or S-Curve speed profile
- ▷ Internal reference clock: 4.9MHz
- ▷ Position pulse setting range (28-bit): -134,217,728 to +134,217,728

I/O Signales

- ▷ Input/Output signals for each axis
- ▷ All I/O signal are optically isolated with 2500Vrms isolation voltage
- ▷ Command pulse output pins: CW and CCW
- ▷ Mechanical limit/switch signal input pins: ±EL, PSD/MSD, and ORG
- ▷ Digital input/output: DIN & DOUT
- ▷ Simultaneous Start/Stop signal: STA and STP

General Specifications

- ▷ Connectors: 68-pin SCSI-type connector
- ▷ Operating Temperature: 0°C - 50°C
- ▷ Storage Temperature: -20°C - 80°C
- ▷ Humidity: 5 - 85%, non-condensing

Power Consumption

- ▷ Slot power supply (input): +5V DC ±5%, 900mA max
- ▷ External power supply (input): +24V_{DC} ±5%, 500mA max
- ▷ External power supply (output): +5V_{DC} ±5%, 500mA, max

PCI-8144 Dimensions

- ▷ 159mm(L) X 126 mm(W)

1.3 Supported Software

1.3.1 Programming Library

Windows 2000/XP DLLs are provided for the PCI-8144. These function libraries are shipped with the board.

1.3.2 MotionCreatorPro

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

Refer to Chapter 5 for more details.

2 Installation

This chapter describes how to install PCI-8144. Please follow these steps below:

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

2.1 Package Contents

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

- PCI-8144: 4-Axis Stepper Motion Control Card
- ADLINK All-in-one Compact Disc
- An optional terminal board for wiring purposes if a different model is ordered.

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

2.2 PCI-8144 Outline Drawing

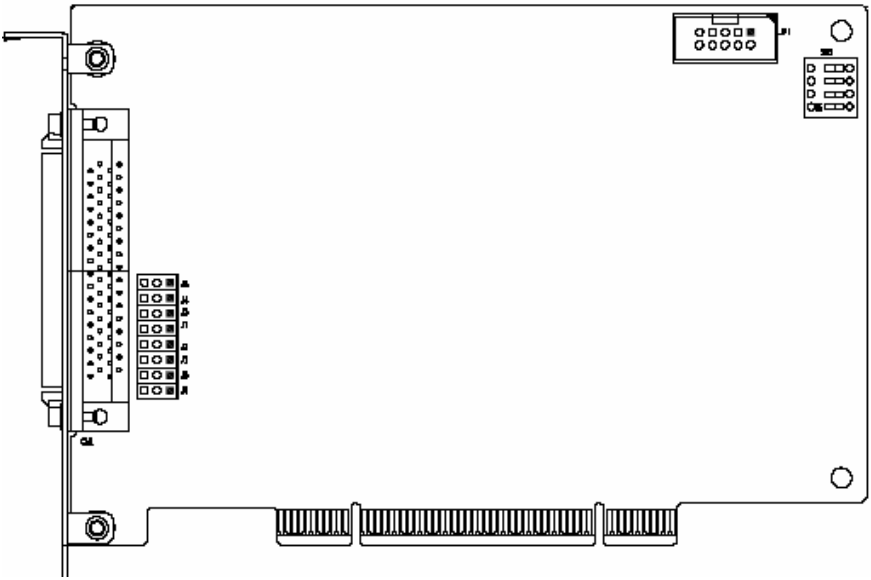


Figure 2-1: PCB Layout of the PCI-8144

- CN1: Input/Output signal connector
- JP2: Direct control to stepper
- SW1: Card ID selection
- J1-J8: Pulse output selection jumper

2.3 PCI-8144 Hardware Installation

2.3.1 Hardware configuration

The PCI-8144 is fully Plug-and-Play compliant. Hence, memory allocation (I/O port locations) and IRQ channel of the PCI card are assigned by the system BIOS. The address assignment is done on a board-by-board basis for all PCI cards in the system.

2.3.2 PCI slot selection

Some computer system may have both PCI and ISA slots. Do not force the PCI card into a PC/AT slot. The PCI-8144 can be used in any PCI slot.

2.3.3 Installation Procedures

1. Read through this manual and setup the jumper according to your application
2. Turn off your computer. Turn off all accessories (printer, modem, monitor, etc.) connected to computer. Remove the cover from the computer.
3. Select a 32-bit PCI expansion slot. PCI slots are shorter than ISA or EISA slots and are usually white or ivory.
4. Before handling the PCI-8144, discharge any static buildup on your body by touching the metal case of the computer. Hold the edge of the card and do not touch the components.
5. Position the board into the PCI slot you have selected.
6. Secure the card in place at the rear panel of the system unit using screws removed from the slot.

2.3.4 Troubleshooting:

If your system doesn't boot or if you experience erratic operation with your PCI board in place, it's most likely caused by an interrupt conflict (possibly an incorrect ISA setup). 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.

2.4 Software Driver Installation

1. Auto run the ADLINK All-In-One CD. Choose Driver Installation -> Motion Control -> PCI-8144.
2. Follow the procedures of the installer.
3. After setup installation is completed, restart windows.

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

2.5 SW1 Card Index Selection

The SW1 switch is used to set the card index. For example, if 1 is set to ON and the others are OFF, that card index is 1. The index value can be from 0 to 15. Refer to the following table for details.

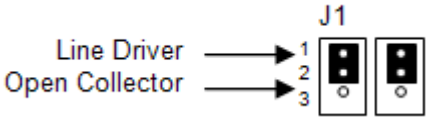
Card ID	Switch Setting (ON=1)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Table 2-1: SW1 Card Index

2.6 J1-J8 Pulse Output Type: Differential / Open Collector

Jumpers J1-J8 are used to set the type of pulse output signals. The output signal type can either be differential line driver or open collector output. Refer to section 3.1 for detail jumper settings.

J1 & J2	Axis 0
J3 & J4	Axis 1
J5 & J6	Axis 2
J7 & J8	Axis 3



2.7 JP2 Direct Control to Stepper

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

Table 2-2: JP2 Direct Control to Stepper

2.8 CN1 Pin Assignments: Main connector

CN1 is the major connector for the motion control I/O signals.

No.	Name	I/O	Function	No.	Name	I/O	Function
1	VDD	O	Isolated +5V Output	35	VDD	O	Isolated +5V Output
2	EGND	-	Ext. power ground	36	EGND	-	Ext. power ground
3	CW+	O	Positive pulse (+)	37	CW+	O	Positive pulse (+)
4	CW-	O	Positive pulse (-)	38	CW-	O	Positive pulse (-)
5	CCW+	O	Negative pulse (+)	39	CCW+	O	Negative pulse (+)
6	CCW-	O	Negative pulse (-)	40	CCW-	O	Negative pulse (-)
7	PEL0	I	Positive end limit signal	41	PEL2	I	Positive end limit signal
8	MEL0	I	Negative end limit signal	42	MEL2	I	Negative end limit signal
9	PSD0	I	Positive slow down signal	43	PSD2	I	Positive slow down signal
10	MSD0	I	Negative slow down signal	44	MSD2	I	Negative slow down signal
11	ORG0	I	Origin signal	45	ORG2	I	Origin signal
12	EGND	-	Ext. power ground	46	EGND	-	Ext. power ground
13	CW+	O	Positive pulse (+)	47	CW+	O	Positive pulse (+)
14	CW-	O	Positive pulse (-)	48	CW-	O	Positive pulse (-)
15	CCW+	O	Negative pulse (+)	49	CCW+	O	Negative pulse (+)
16	CCW-	O	Negative pulse (-)	50	CCW-	O	Negative pulse (-)
17	PEL1	I	Positive end limit signal	51	PEL3	I	Positive end limit signal
18	MEL1	I	Negative end limit signal	52	MEL3	I	Negative end limit signal
19	PSD1	I	Positive slow down signal	53	PSD3	I	Positive slow down signal
20	MSD1	I	Negative slow down signal	54	MSD3	I	Negative slow down signal
21	ORG1	I	Origin signal	55	ORG3	I	Origin signal
22	STP/EMG	I	Forced stop signal	56	STA	I	External start signal
23	DIN0	I	Digital Input 0	57	DOUT0	O	Digital Output 0
24	DIN1	I	Digital Input 1	58	DOUT1	O	Digital Output 1
25	DIN2	I	Digital Input 2	59	DOUT2	O	Digital Output 2
26	DIN3	I	Digital Input 3	60	DOUT3	O	Digital Output 3
27	DIN4	I	Digital Input 4	61	DOUT4	O	Digital Output 4
28	DIN5	I	Digital Input 5	62	DOUT5	O	Digital Output 5
29	DIN6	I	Digital Input 6	63	DOUT6	O	Digital Output 6
30	DIN7	I	Digital Input 7	64	DOUT7	O	Digital Output 7
31	VDD	-	Isolated +5V Output	65	DO_COM	-	Common for DO
32	VDD	-	Isolated +5V Output	66	DO_COM	-	Common for DO
33	EGND	-	Ext. power ground	67	EGND	-	Ext. power ground
34	EX+24V	I	+24V isolation power input	68	EX+24V	I	+24V isolation power input

Table 2-3: CN1 Pin Assignments

3 Signal Connections

Signal connections of all I/O's are described in this chapter. Refer to the contents of this chapter before wiring any cables between the PCI-8144 and any stepper drivers.

This chapter contains the following sections:

3.1 Pulse Output Signals CW and CCW

There are 4 axis pulse output signals on the PCI-8144. For each axis, two pairs of CW and CCW signals are used to transmit the pulse train and to indicate the direction. In this section, the electrical characteristics of the CW and CCW signals are detailed. Each signal consists of a pair of differential signals. For example, CW2 consists of CW2+ and CW2- signals. The following table shows all pulse output signals on CN1.

CN1 Pin No.	Signal Name	Description	Axis #
3	CW0+	Pulse signals (+)	1
4	CW0-	Pulse signals (-)	1
5	CCW0+	Direction signal (+)	1
6	CCW0-	Direction signal (-)	1
13	CW1+	Pulse signals (+)	2
14	CW1-	Pulse signals (-)	2
15	CCW1+	Direction signal (+)	2
16	CCW1-	Direction signal (-)	2
17	CW2+	Pulse signals (+)	3
18	CW2-	Pulse signals (-)	3
39	CCW2+	Direction signal (+)	3
40	CCW2-	Direction signal (-)	3
47	CW3+	Pulse signals (+)	4
48	CWT3-	Pulse signals (-)	4
49	CCW3+	Direction signal (+)	4
50	CCW3-	Direction signal (-)	4

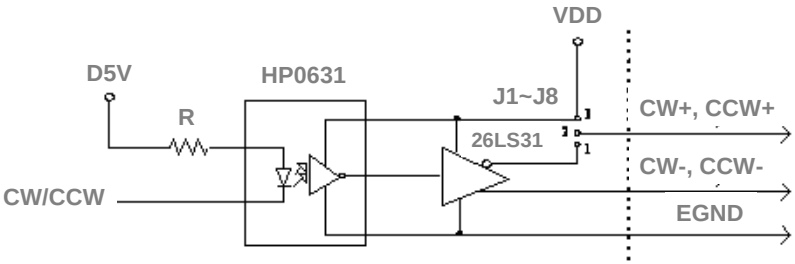
The output of the CW or CCW signals can be configured by jumpers as either differential line drivers or open collector output. Users can select the output mode either by closing breaks between 1 and 2 or 2 and 3 of jumpers J1-J8 as follows:

Output Signal	For differential line driver output, close breaks between 1 and 2 of:	For open collector output, close breaks between 2 and 3 of:
CW0-	J1	J1
CCW0-	J2	J2
CW1-	J3	J3
CCW1-	J4	J4
CW2-	J5	J5
CCW2-	J6	J6
CW3-	J7	J7
CCW3-	J8	J8

The **default** setting of CW and CCW is set to differential line driver mode.

The following wiring diagram is for CW and CCW signals on the 4 axes.

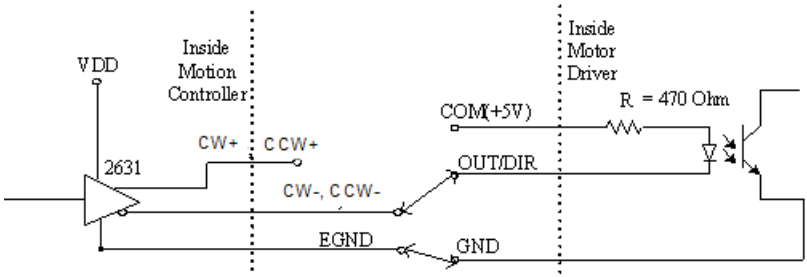
PCI-8144:



Note: If the pulse output is set to open collector output mode, CW- and CCW- are used to transmit CW and CCW signals. The sink current must not exceed 20mA on the CW- and CCW- pins. The default setting of jumper is 1-2 shorted.

Suggest Usage: Jumper 2-3 shorted and connect CW+/CCW+ to a 470 ohm pulse input interface's COM of driver. See the following figure.

Choose one of CW/CCW+ and CW/CCW- to connect to driver's OUT/DIR



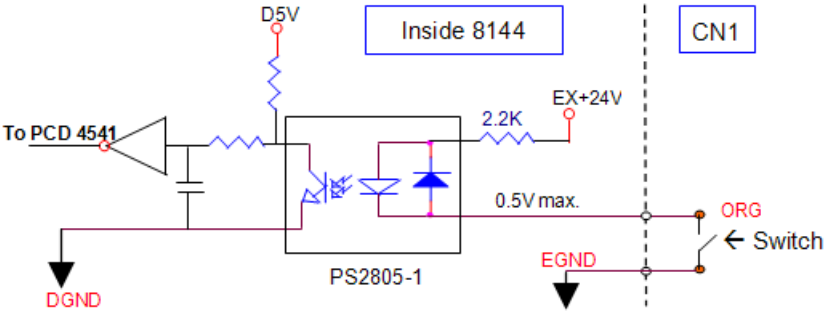
Warning: **The sink current must not exceed 20mA or the 2631 will be damaged!**

3.2 Origin Signal ORG

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

CN1 Pin No	Signal Name	Axis #
11	ORG0	1
21	ORG1	2
45	ORG2	3
55	ORG3	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 +24V @ 10mA minimum. An internal filter circuit is used to filter out any high frequency spikes, which may cause errors in the operation.



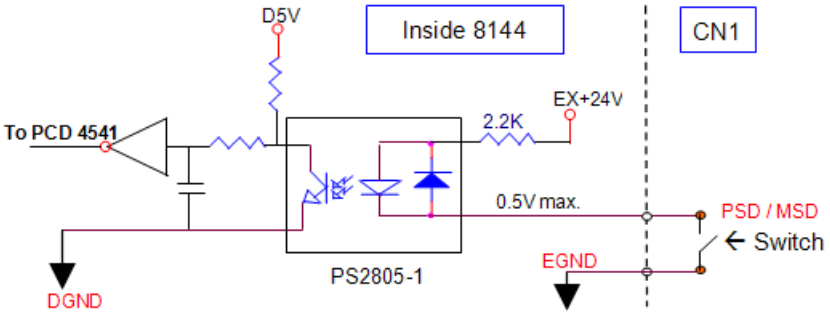
When the motion controller is operated in the home return mode, the ORG signal is used to inhibit the control output signals (CW and CCW). For detailed operations of the ORG signal, refer to Section 4.

3.3 Slow Down input signal

The PCI-8144 provides slow down function through SD input pin. The signal names, pin numbers, and axis numbers are shown in the following table:

CN1 Pin No	Signal Name	Axis #	CN1 Pin No	Signal Name	Axis #
9	PSD0	1	43	PSD2	3
10	MSD0	1	44	MSD2	3
19	PSD1	2	53	PSD3	4
20	MSD1	2	54	MSD3	4

The SD input pin wiring diagram is as follows:

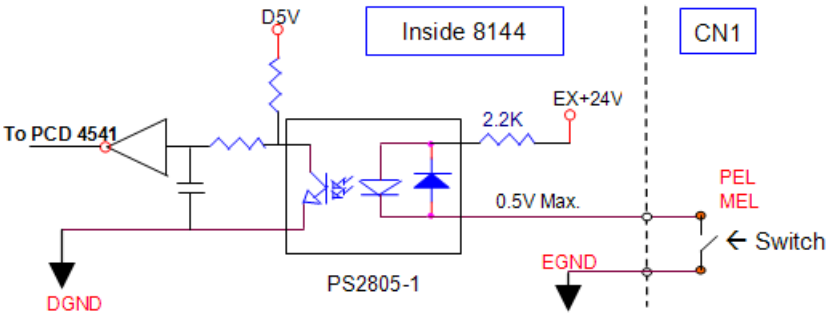


3.4 End-Limit Signals PEL and MEL

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:

CN1 Pin No	Signal Name	Axis #	CN1 Pin No	Signal Name	Axis #
7	PEL0	1	41	PEL2	3
8	MEL0	1	42	MEL2	3
17	PEL1	2	51	PEL3	4
18	MEL1	2	52	MEL3	4

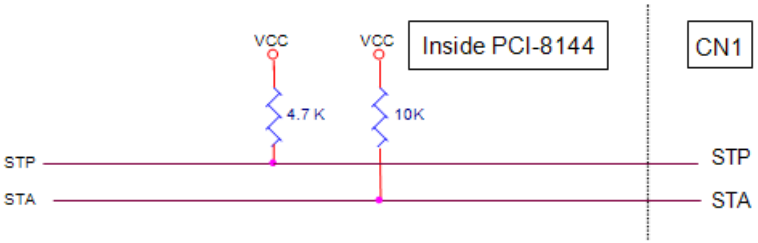
A circuit diagram is shown in the diagram below. The external limit switch should have a contact capacity of +24V @ 10mA minimum. EL logical can be configured by registers controlling. For more details on EL operation, refer to Section 4.



3.5 Simultaneously Start/Stop Signals STA and STP

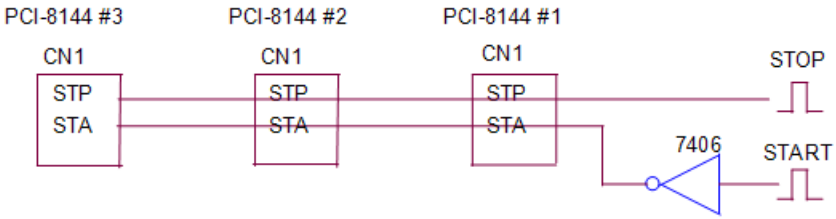
The PCI-8144 provides STA and STP signals, which enable simultaneous start/stop of motions on multiple axes. The STA and STP signals are on CN1, STP @ pin 22 and STA @ pin 56.

The diagram below shows the onboard circuit. The STA and STP signals of the four axes are tied together respectively.



The STP and STA signals are input signals. To operate the start and stop action simultaneously, both software control and external control are needed. With software control, the signals must be generated from external event to the chip of PCD4541. Users 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-8144 cards, tie all STA and all STP signals of all cards resepectively for simultaneous start/stop control on all concerned axes. In this case, connect CN1 as below:



3.6 Termination Board

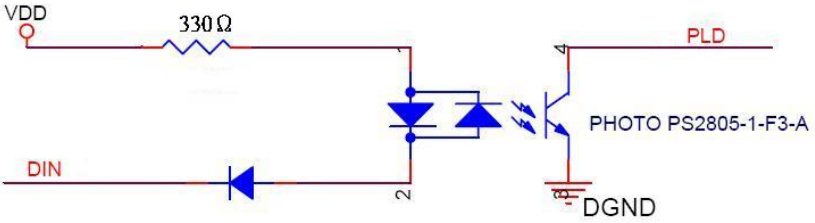
CN1 of the PCI-8144 can be connected with a DIN-68S, including the ACL-10569-1 cable (a 68-pin SCSI-SCSI cable). The DIN-68S is a general purpose 68-pin SCSI-II DIN-socket. It has easy wiring screw terminals and an easily installed DIN socket that can be mounted onto the DIN rails

3.7 General Purpose DIO

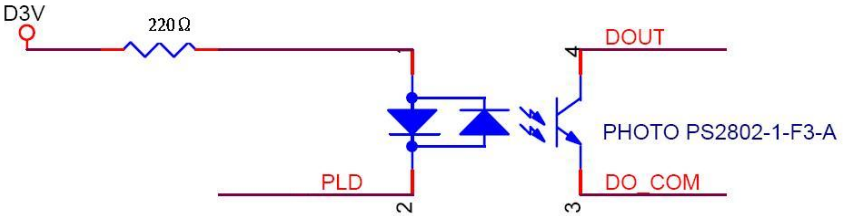
PCI-8144 has 8 opto-isolated digital outputs and 8 open collector digital inputs for general purpose use. Pin assignments are illustrated in the table below:

CN1 Pin No	Signal Name	CN1 Pin No	Signal Name
23	DIN0	57	DOUT0
24	DIN1	58	DOUT1
25	DIN2	59	DOUT2
26	DIN3	60	DOUT3
27	DIN4	61	DOUT4
28	DIN5	62	DOUT5
29	DIN6	63	DOUT6
30	DIN7	64	DOUT7
		65	DO_COM
		66	DO_COM

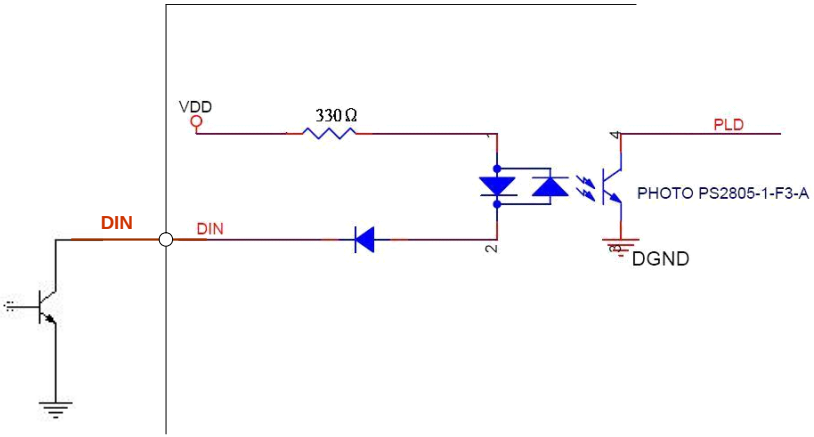
3.7.1 Isolated Input channels



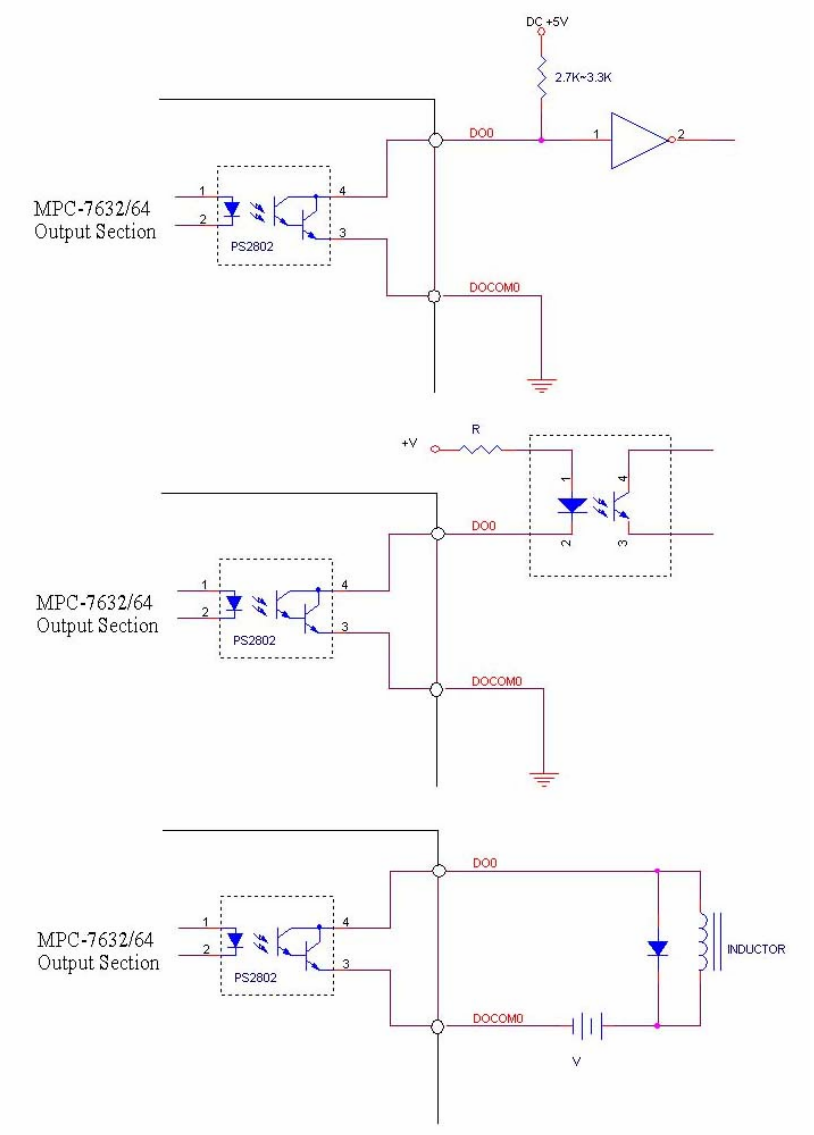
3.7.2 Isolated Output channels



3.7.3 Example of input connection



3.7.4 Example of output connection



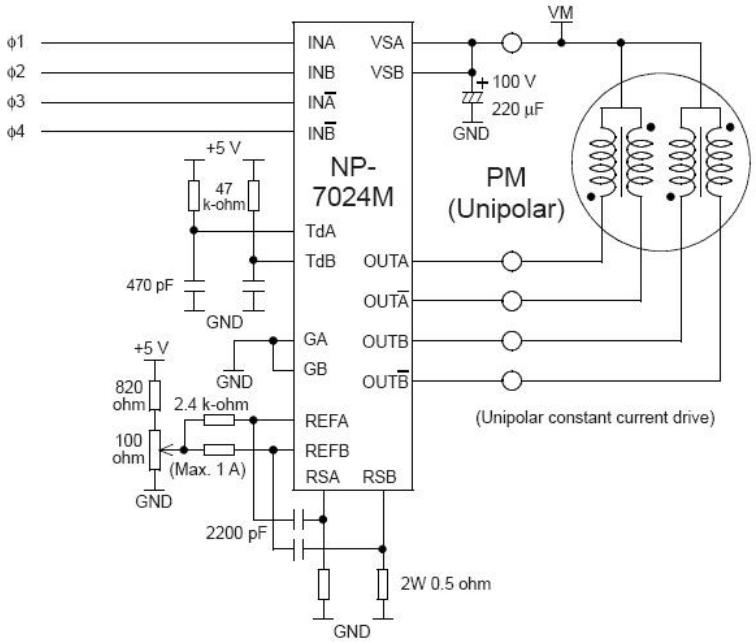
3.8 JP2 pin define for stepping singal (Optional)

PCI-8144 provides another optional JP2 pins for stepping motor signal. Pin definitions of JP2 and connection example are described as follow:

JP2 Pin No	Signal Name	JP2 Pin No	Signal Name
1	Φ1x	2	Φ2x
3	Φ3x	4	Φ4x
5	Φ1y	6	Φ2y
7	Φ3y	8	Φ4y
9	Φ1z	10	Φ2z
11	Φ3z	12	Φ4z
13	Φ1u	14	Φ2u
15	Φ3u	16	Φ4u

Pins 17 to 26 are not used

Connection example:



4 Operation Theory

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

Section 4.1: Classifications of Motion Controller
Section 4.2: Motion Control Modes
Section 4.3: Motor Driver Interface
Section 4.4: Mechanical switch Interface
Section 4.5: The Counters
Section 4.6: The Comparators
Section 4.7: Other Motion Functions
Section 4.8: Interrupt Control
Section 4.9: Multiple Cards Operation

4.1 Classifications of Motion Controller

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

4.1.1 Voltage motion control interface

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

4.1.2 Pulse motion control interface

The second interface of motion and motor control is a pulse train type. As a trend of digital world, pulse trains represent a new concept to motion control. The counts of pulses show how many steps of a motor rotates and the frequency of pulses show how fast a motor runs. The time duration of frequency changes represent the acceleration rate of a motor. Because of this interface, a servo or stepper motor can be easier than an analog type for positioning applications. It means that motion and motor control can be separated more easily by this way.

Both of these two interfaces need to provide for gains tuning. For analog position controllers, the control loops are built inside and users must tune the gain from the controller. For pulse type position controllers, the control loops are built outside on the motor drivers and users must tune the gains on drivers.

For more than one axes' operation, motion control seems more important than motor control. In industial applications, reliable is a very important factor. Motor driver vendors make good performance products and a motion controller vendors make powerful and variety motion software. Integrated two products make our machine go into perfect.

4.1.3 Network motion control interface

Recently, there was a new control interface was introduced--a network motion controller. The command between motor driver and motion controller is not analog or pulses signal any more. It is a network packet which contents position information and motor information. This type of controller is more reliable because it is digitized and packetized. Because a motion controller must be real-time, the nerowrk must have real-time capacity around a cycle time below 1 mini-second. This means that non-commercial networks cannot do this job. It must have a specific network, such as Mitsubishi SSCNET. The network may also be built with fiberoptics to increase communication reliability.

4.1.4 Software real-time motion control kernel

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

A motion control system needs an absolutely real-time control cycle and the calculation on controller must provide a control data at the same cycle. If not, the motor will not run smoothly. Many machine makers will use PC's computing power to do this. A feedback counter card can simply be used and a voltage output or pulse output card to make it. This method is very low-end and takes much software effort. For sure their realtime performance, they will use a real-time software on the system. It increases the complexity of the system too. But this method is the most flexible way for a professional motion control designers. Most of these methods are on NC machines.

4.1.5 DSP motion control kernel

A DSP-based motion controller kernel solves real-time software problem on computer. DSP is a micro-processer itself and all motion control calculations can be done on it. There is no real-time software problem because DSP has its own OS to arrange all the procedures. There is no interruption from other inputs or context switching problem like Windows based computer. Although it has such a perfect performance on real-time requirements, its calculation speed is not as fast as PC's CPU at this age. The software interfacing between DSP controller's vendors and users is not easy to use. Some controller vendors provide some kind of assembly languages for users to learn and some controller vendors provide only a handshake documents for users to use. Both ways are not easy to use. DSP based controller provide a better way than software kernel for machine makers to build they applications.

4.1.6 ASIC motion control kernel

An ASIC motion control kernel is falls between software kernel and DSP kernel in terms of difficulty. It has no real-time problem because all motion functions are done via the ASIC. Users or controller's vendors just need to set some parameters which the ASIC requires and the motion control will be done easily. This kind of

motion control separates all system integration problems into 4 parts: Motor driver's performance, ASIC outputting profile, vendor's software parameters to the ASIC, and users' command to vendors' software. It makes motion controller co-operated more smoothly between devices.

4.1.7 Compare Table of all motion control types

	Software	ASIC	DSP
Price	Fair	Cheap	Expensive
Functionality	Highest	Low	Normal
Maintenance	Hard	Easy	Fair

	Analog	Pulses	Network
Price	High	Low	Normal
Signal Quality	Fair	Good	Reliable
Maintenance	Hard	Easy	Easy

4.1.8 PCI-8144 motion controller type

The PCI-8144 is an ASIC based, pulse type motion controller made into three blocks: motion ASIC, PCI card, software motion library. The motion ASIC can be accessed via our software motion libray under many kinds of Windows NT/2000/XP, Linux, and RTX driver. Our software motion library provides one-stop-function for controlling motors. All the speed parameter calculations are done via our library.

For example, to perform a one-axis point to point moition with a trapezoidal speed profile, only fill the target position, speed, and acceleration time in one function. Then the motor will run as the profile. It takes no CPU resources because every control cycle pulse generation is done by the ASIC. The precision of target position depends on motor drivers' closed loop control performance and mechnical parts, not on motion controller's command because the motion controller is only responsible for sending correct pulses counts via a desired speed profile. So it is much easier for programmers, mechnical or electrical engineers to find out problems.

4.2 Motion Control Modes

Motion control makes the motors run according to a specific speed profile, path trajectory and synchronous condition with other axes. The following sections describe the motion control modes of this motion controller could be performed.

4.2.1 Coordinate system

The Cartesian coordinate is used and pulses are in the unit of length. The physical length depends on mechanical parts and motor's resolution. For example, if a motor is on a screw ball, and the pitch of screw ball is 10mm and the pulses needed for a round of motor are 10,000 pulses. One pulse's physical unit is equal to 10mm/10,000p =1 mm.

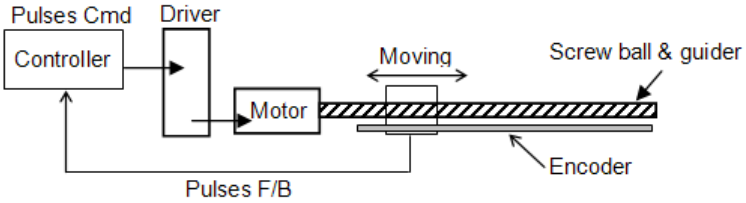
Just set a command with 15,000 pulses for motion controller if we want to move 15mm. How about if we want to move 15.0001mm? **Simple! The motion controller will keep the residue value less than 1 pulse and add it to next command.**

The motion controller sends incremental pulses to motor drivers. It means that we can only send relative command to motor driver. But we can solve this problem by calculating the difference between current position and target position first. Then send the differences to motor driver. For example, if current position is 1000 and we want to move a motor to 9000, you can use an absolute command to set a target position of 9000. Inside the motion controller, it will get current position 1000 first then calculate the difference from target position. The result is +8000. So, the motion controller will send 8000 pulses to motor driver to move the position of 9000.

Sometimes, users need to install a linear scale or external encoder to check machine's position. But how do you to build this coordinate system? If the resolution of external encoder is 10,000

pulses per 1mm and the motor will move 1mm if the motion controller send 1,000 pulses, It means that when we want to move 1 mm, we need to send 1,000 pulses to motor driver then we will get the encoder feedback value of 10,000 pulses. If we want to use an absolute command to move a motor to 10,000 pulses position and current position read from encoder is 3500 pulses, how many pulses will it send to motor driver? The answer is (10000 – 3500) / (10,000 / 1,000)=650 pulses. The motion controller will calculate it automatically if users set “move ratio” already. The “move ratio” means the (feedback resolution/command resolution)

Note: The PCI-8144 provides only one dimension motion function.



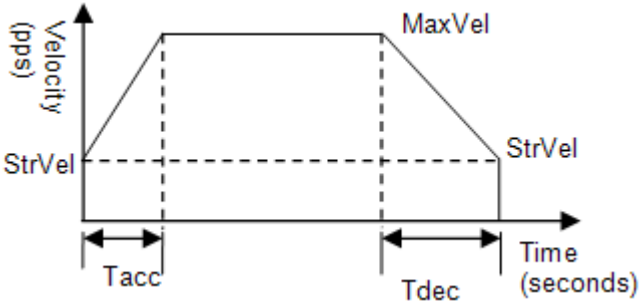
4.2.2 Absolute and relative position move

In the coordinate system, we have two kinds command for users to locate the target position. One is absolute and the other is relative. Absolute command means that user give the motion controller a position, then the motion controller will move a motor to that position from current position. Relative command means that user give the motion controller a distance, then the motion controller will move motor by the distance from current position. During the movement, users can specify the speed profile. It means user can define how fast and at what speed to reach the position.

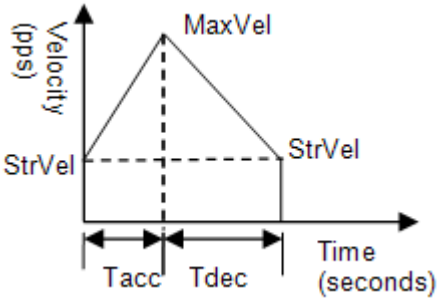
The PCI-8144 provides only relative position move because it is designed for stepper motor. It is meaningless for a stepper motor controller to have the encoder feedback interface for absolute motion.

4.2.3 Trapezoidal speed profile

Trapezodial speed profile means the acceleration/deceleration area follows a 1st order linear velocity profile (constant acceleration rate). The profile chart is shown as below:



The area of the velocity profile represents the distance of this motion. Sometimes, the profile looks like a triangle because the desired distance from user is smaller than the area of given speed parameters. When this situation happens, the motion controller will lower the maximum velocity but keep the acceleration rate to meet user's distane requirement. The chart of this situation is shown as below:



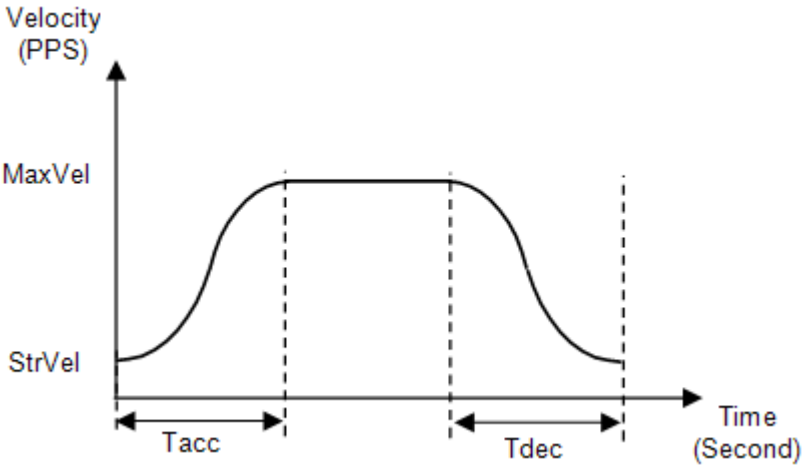
This kind of speed profile could be applied on velocity mode, position mode in one axis or multi-axes linear interpolation and two axes circular interpolation modes.

The PCI-8144 Tacc and Tdec are always the same.

4.2.4 S-curve and Bell-curve speed profile

S-curve means the speed profile in accelerate/decelerate area follows a 2nd order curve. It can reduce vibration at the beginning of motor start and stop. In order to speed up the acceleration/deceleration during motion, we need to insert a linear part into these areas. We call this shape as “Bell” curve. It adds a linear curve between the upper side of s-curve and lower side of s-curve. This shape improves the speed of acceleration and also reduces the vibration of acceleration.

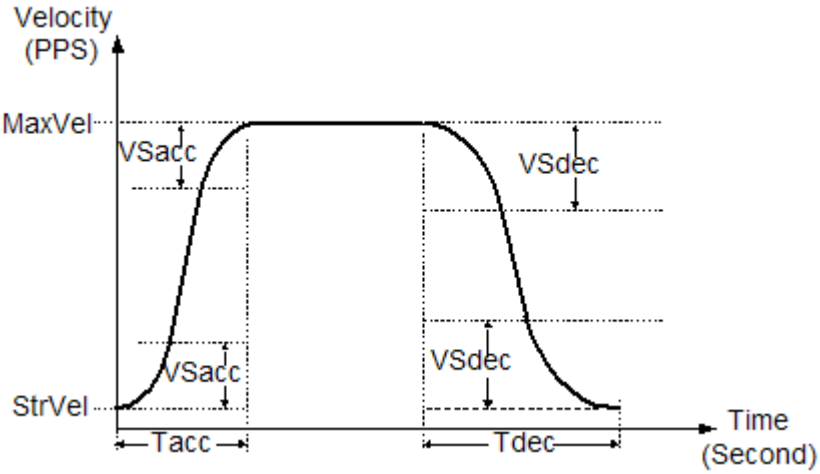
a) For a purse S-curve, we define its shape’s parameter as below:



- Tacc: Acceleration time in second
- Tdec: Deceleration time in second
- StrVel: Starting velocity in PPS
- MaxVel: Maximum velocity in PPS

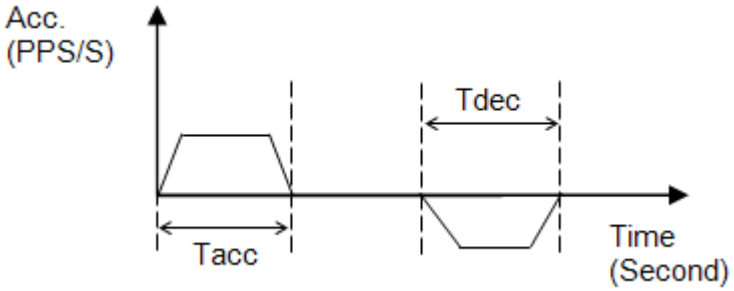
The PCI-8144 Tacc and Tdec are always the same.

b) For a bell curve, we define its shape parameters as below: (the PCI-8144 dosen't support this mode)



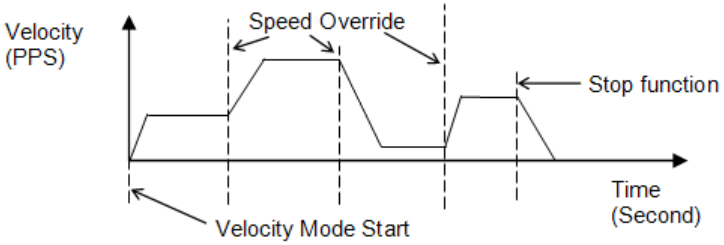
- Tacc: Acceleration time in second
- Tdec: Deceleration time in second
- StrVel: Starting velocity in PPS
- MaxVel: Maximum velocity in PPS
- VSacc: S-curve part of a bell curve in deceleration in PPS
- VSdec: S-curve part of a bell curve in deceleration in PPS

If VSacc or VSdec=0, it means acceleration or deceleration use pure S-curve without linear part. The Acceleration chart of bell curve is shown below:



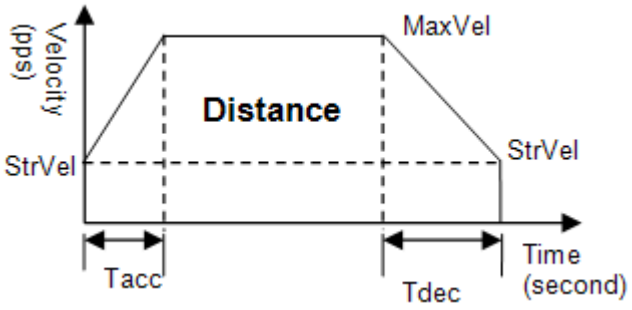
4.2.5 Velocity mode

Velocitiy mode means the pulse command is continuously outputing until a stop command is issued. The motor will run without a target position or desired distance unless it is stopped by other reasons. The output pulse accelerates from a starting velocity to a specified maximum velocity. It can be follow a linear or S-curve acceleration shape. The pulse output rate is kept at maximum velocity until another velocity command is set or a stop command is issued. The velocity could be overrided by a new speed setting. Notice that the new speed could not be a reversed speed of original running speed. The speed profile of this kind of motion is shown as below:



4.2.6 One axis position mode

Position mode means the motion controller will output a specific amount of pulses which is equal to users' desired position or distance. The unit of distance or position is pulse internally on the motion controller. The minimum length of distance is one pulse. However, in PCI-8144, we provide a floating point function for users to transform a physical length to pulses. Inside our software library, we will keep those distance less than one pulse in register and apply them to the next motion function. Besides positioning via pulse counts, our motion controller provides three types of speed profile to accomplish positioning. There are 1st order trapezoidal, 2nd order S-curve, and mixed bell curve. Users can call respective functions to perform that. The following char shows the relationship between distance and speed profile. We use trapezoidal shape to show it.



The distance is the area of the V-t diagram of this profile.

The PCI-8144 Tacc and Tdec are always the same.

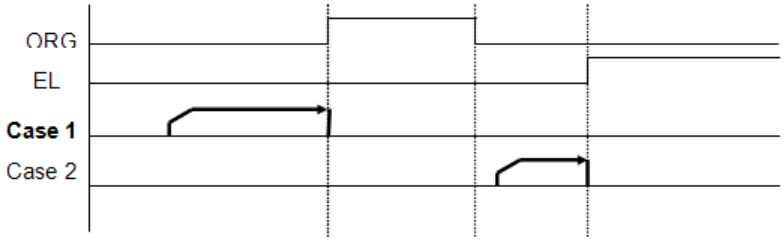
4.2.7 Home Return Mode

Home return means searching a zero position point on the coordinate. Sometimes, users use ORG pin as a zero position on the coordinate. At the beginning of machine power on, the program needs to find a zero point of this machine. Our motion controller provides following functions to make it.

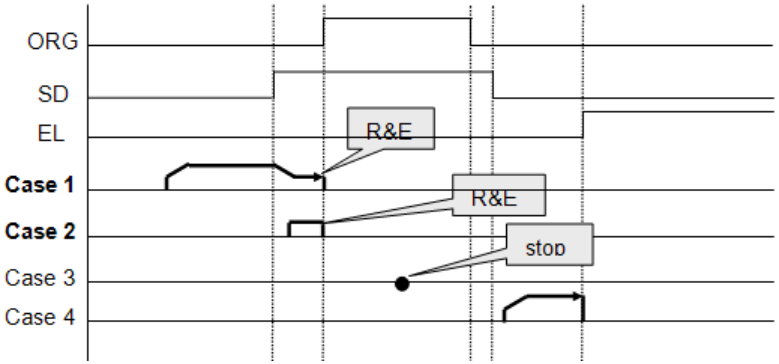
```
_8144_enable_sd_signal
_8144_enable_org_stop,
_8144_tv_move,
_8144_sv_move.
```

When ORG signal control is enabled (home return operation), the ORG signal is turned ON and the motor will stop immediately. After that, if the ORG signal goes OFF, the motor will remain stopped. Those signal statues can be used by the function, "_8144_get_mio_status".

- When SD is not installed



- When SD is installed and SD is not latched

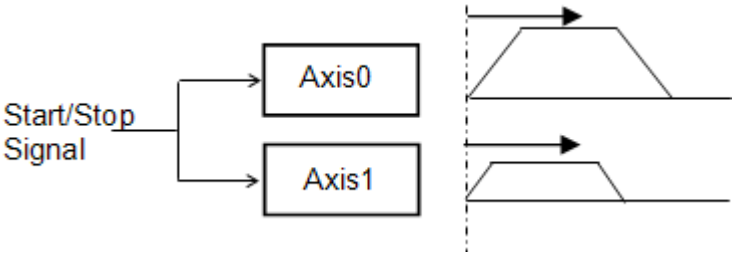


Example.

```
_8144_enable_sd_signal( AxisNo, Enable); //Enable
     SD signal control
_8144_enable_org_stop( AxisNo, Enable); //Enable
     ORG signal stop
_8144_sv_move(AxisNo, StrVel, MaxVel, Tacc); //
     perform a sv-move
```

4.2.8 Synchronous Start Function

Synchronous motion means more than one axis can be started by a synchronous signal (STA) which could be external or internal signals. For external signal, users must set move parameters first for all axes then these axes will wait an extern start/stop command to start or stop. For internal signal, the start command could be from a software start function. Once it is issued, all axes which are in waiting synchronous mode will start at the same time.



The PCI-8144 card suports external start function. When the user sets the external start by _8144_external_start function for each axis which the user want to perform external start, then set the move function for those axes. Thoes axis will start at the same time when STA signal goes ON.

4.2.9 Auto Home Move Mode

Depending on the software homing mode design, the PCI-8144 offers an auto homing move function which means the axis will move to a zero position point on the coordinate. This mode is used to add an auto home moving function on the normal home return mode described in the previous section no matter which position the axis is. The following diagram shows an example for normal home mode usage for which the start point is between the ORG and EL signal or exists in the ORG signal. The ORG offset can't be zero because the motion ASIC of the PCI-8144 offers a level trigger for the ORG signal. The suggested value is the double length of the ORG area. A homing status function can be used to monitor the current status of the homing operation.

Case 1: SD (Slow down) > Stop at ORG

Case 2: EL (End Limit) > SD (Slow down) > Stop at ORG

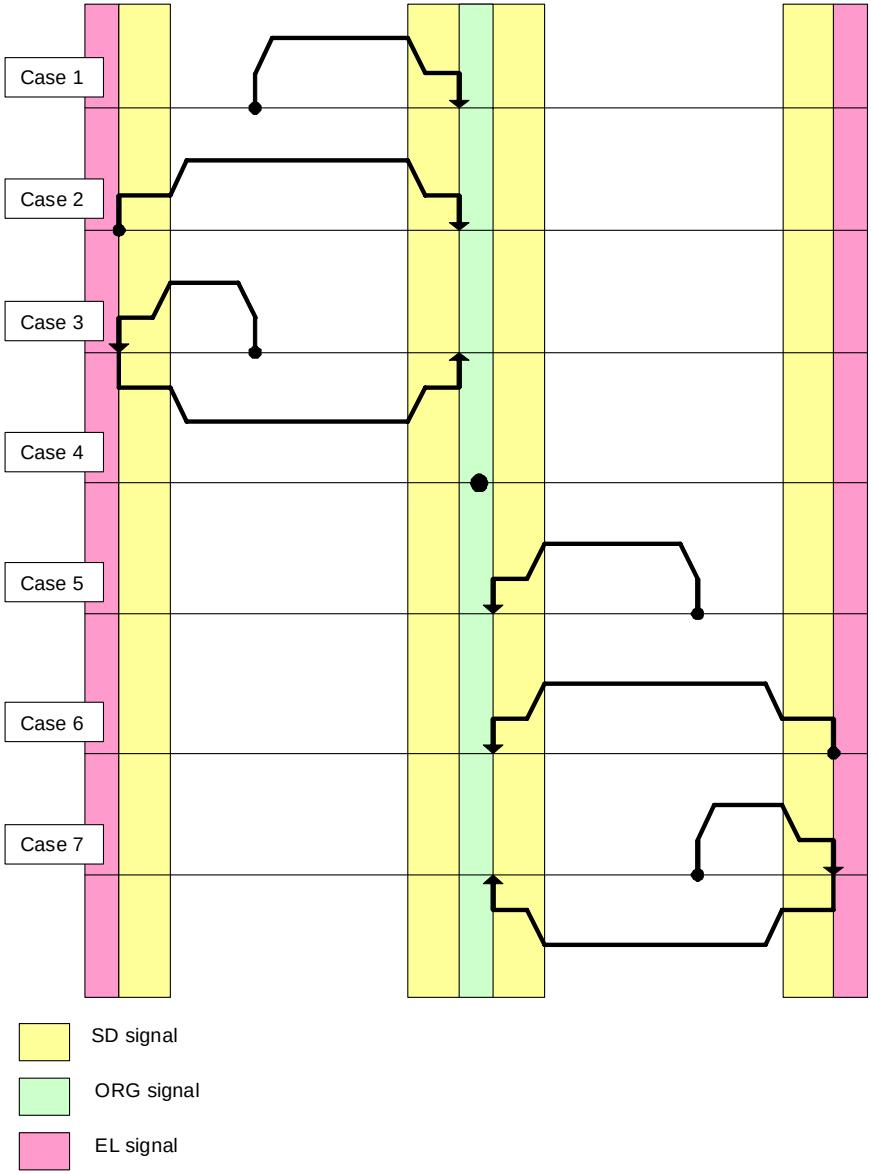
Case 3: Inversed direction moving > EL (End Limit) > Return > SD (Slow down) > Stop at ORG

Case 4: Stop at ORG

Case 5: SD (Slow down) > Stop at ORG

Case 6: EL (End Limit) > SD (Slow down) > Stop at ORG

Case 7: Inversed direction moving > EL (End Limit) > Return > SD (Slow down) > Stop at ORG


Case 1
Case 2
Case 3
Case 4
Case 5
Case 6
Case 7
SD signal
ORG signal
EL signal

4.3 The motor driver interface

We provide several dedicated I/Os which can be connected to motor driver directly and have their own functions. Motor drivers have many kinds of I/O pins for external motion controller to use. We classify them to two groups. One is pulse I/O signals including pulse command and encoder interface. The other is digital I/O signals including servo ON, alarm, INP, servo ready, alarm reset and emergency stop inputs. The following sections will describe the functions these I/O pins.

The PCI-8144 has only pulse command interface.

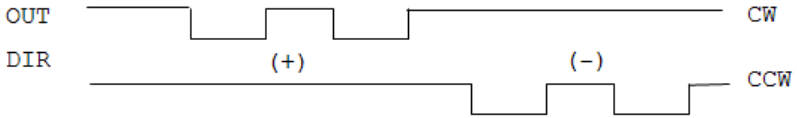
4.3.1 Pulse Command Output Interface

The motion controller uses pulse command to control servo/stepper motors via motor drivers. Please set the drivers to position mode which can accept pulse trains as position command. The pulse command consists of two signal pairs. It is defined as CW and CCW pins on connector. Each signal has two pins as a pair for differential output.

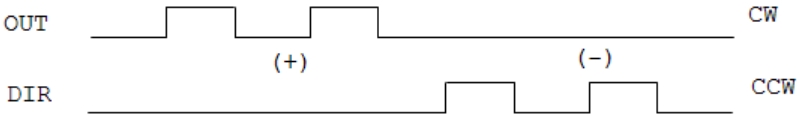
Dual Pulse Output Mode (CW/CCW Mode)

The waveform of the CW and CCW pins represent CW (clockwise) and CCW (counter clockwise) pulse output respectively. The numbers of pulse represent distance in pulse. The frequency of the pulses represents speed in pulse per second. 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 diagram shows the output waveform of positive (+) commands and negative (-) commands.

Pulse outmode = 0: (Pulse is normally high)



Pulse outmode = 1: (Pulse is normally low)



The command pulses are counted by a 24-bit preset countdown counter. The preset counter can store a value of total pulses outputting from controller.

4.4 Mechanical switch interface

We provide some dedicated input pins for mechanical switches like original switch (ORG), plus and minus end-limit switch (±EL), slow down switch (SD). These switches' response time is very fast, only a few ASIC clock times. There is no real-time problem when using these signals. All functions are done by motion ASIC. The software can just do nothing and only need to wait the results.

4.4.1 Original or home signal

Our controller provides one original or home signal for each axis. This signal is used for defining zero position of this axis. The logic of this signal must be set properly before doing home procedure. Please refer to home mode section for details.

4.4.2 End-Limit switch signal

The end-limit switches are usually installed on both ending sides of one axis. We must install plus EL at the positive position of the axis and minus EL at the negative position of the axis. These two signals are for safty reason. If they are installed reversely, the protection will be invalid. Once the motor's moving part touches one of the end-limit signals, the motion controller will stop sending pulses and output an ERC signal. It can prevent machine crash when miss operation.

4.4.3 Slow down switch

The slow down signals are used to force the command pulse to decelerate to the starting velocity when it is active. This signal is used to protect a mechanical moving part under high speed movement toward the mechanism's limit. The SD signal is effective for both plus and minus directions.

4.5 The Counters

4.5.1 Preset Command position counter

The preset command position counter is a 24-bit binary down-count counter. It provides the information of the current command counts which are not outputted.

4.6 Interrupt Control

The motion controller can generate an interrupt signal to the host PC. It is much useful for event-driven software application. There are two kinds of interrupt sources on PCI-8144. One is motion interrupt source and the other is GPIO interrupt sources. Motion and GPIO interrupt sources can be maskable. Motion interrupt sources can be maskable by **`_8144_set_motion_interrupt_factor()`**. Its mask bits are shown as following table:

Motion Interrupt Source Bit Settings (1=Enable,0=Disable)

Bit	Description
0	Motion Stop
1-15	(reserve)

The GPIO interrupt sources are maskable. The mask bits table is shown below:

GPIO Interrupt Source Bit Settings (1=Enable,0=Disable)

Bit	Description
0	DI0 falling edge
1	DI1 falling edge
2	DI2 falling edge
3	DI3 falling edge
4	DI4 falling edge
5	DI5 falling edge
6	DI6 falling edge
7	DI7 falling edge
8	DI0 rising edge
9	DI1 rising edge
10	DI2 rising edge
11	DI3 rising edge
12	DI4 rising edge
13	DI5 rising edge
14	DI6 rising edge
15	DI7 rising edge

After seting the factors of interrupt, users can use wait functions to wait the interrupts, it uses no processor time while waiting for the specified interrupt to become signaled or the time-out interval to elapse.

The steps for using interrupts:

1. Set interrupt sources for Motion or GPIO interrupts.

```
_8144_set_motion_interrupt_facor(AXIS0, 0x01);
     // Axis0 stop
_8144_set_gpio_interrupt_factor(CARD0, 0x01);
     // DI0 falling edge
```

2. Using wait function to wait the specified interrupt

```
_8144_wait_single_motion_interrupt(AXIS0, 0x01,
     1000);
     // Wait 1000ms for normally stop interrupt
_8144_wait_single_gpio_interrupt(CARD0, 0x01,
     1000)
     // Wait 1000ms for DI0 falling edge
     interrupt
```

3. Disable interrupt sources

```
_8144_set_motion_interrupt_facor(AXIS0, 0x0);
_8144_set_gpio_interrupt_factor(CARD0, 0x0);
```

4.7 Multiple Card Operation

The motion controller allows more than one card in one system. Since the motion controller is plug-and-play compatible, the base address and IRQ setting of the card are automatically assigned by the PCI BIOS at the beginning of system booting. Users don't need and can't change the resource settings.

When multiple cards are applied to a system, the number of card must be noted. The card number depends on the card ID switch setting on the the board. The axis number is depends on the card ID. For example, if three motion controller cards are plugged in to PCI slots, and the corresponding card ID is set, then the axis number on each card will be:

	Axis No.	**X**	**Y**	**Z**	**U**
Card ID					
0		0	1	2	3
2		8	9	10	11
3		12	13	14	15

Notice that if there has the same card ID on multiple cards, the function will not work correctly.

5 MotionCreatorPro

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 8144 cards to verify correct operation. The MotionCreatorPro software provides a simple yet powerful means to setup, configure, test, and debug a motion control system that uses 8144 cards.

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

5.1 Run MotionCreatorPro

After installing the software drivers for the 8144 in Windows 2000/XP, the MotionCreatorPro program can be located at <chosen path> \PCI-Motion\MotionCreatorPro. To execute the program, double click on the executable file or use Start>Program Files>PCI-Motion>MotionCreatorPro.

5.2 About MotionCreatorPro

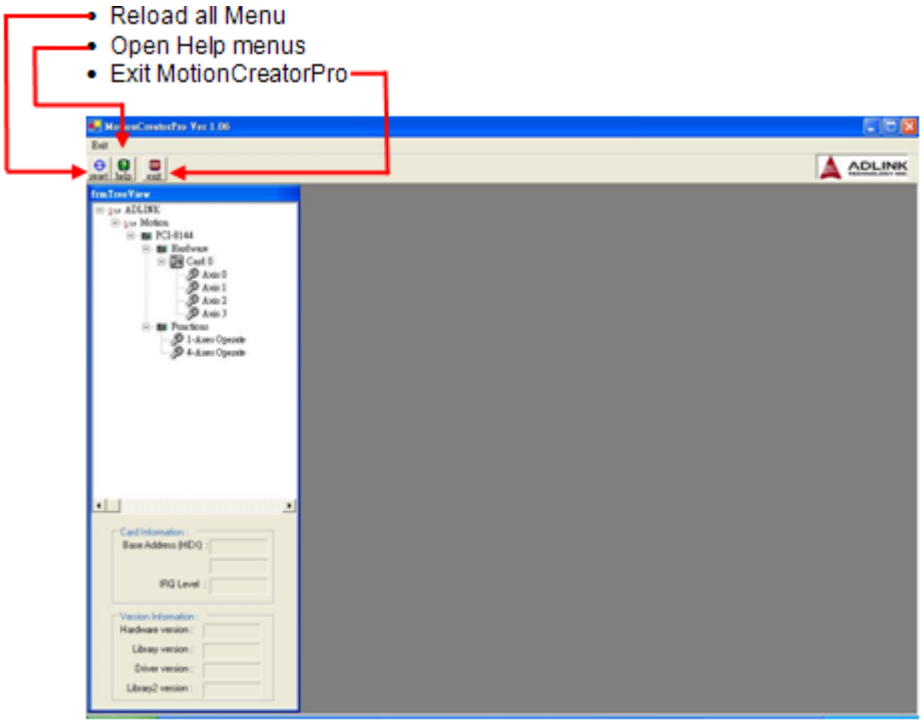
Before Running MotionCreatorPro, the following issues should be kept in mind.

1. MotionCreatorPro is a program written in VB.NET 2003, and is available only for Windows 2000/XP with a screen resolution higher than 1024x768. It cannot be run under DOS.
2. MotionCreatorPro allows users to save settings and configurations for 8144 cards. Saved configurations will be automatically loaded the next time MotionCreatorPro is executed. Two files, **`8144.ini`** and **`8144MC.ini`**, in the **windows root directory** are used to save all settings and configurations.
3. To duplicate configurations from one system to another, copy **`8144.ini`** and **`8144MC.ini`** into the windows root directory.
4. If multiple 8144 cards use the same MotionCreatorPro saved configuration files, the DLL function call **`_8144_config_from_file()`** can be invoked within a user developed program. This function is available in a DOS environment as well.

5.3 MotionCreatorPro Form Introducing

5.3.1 Main Menu

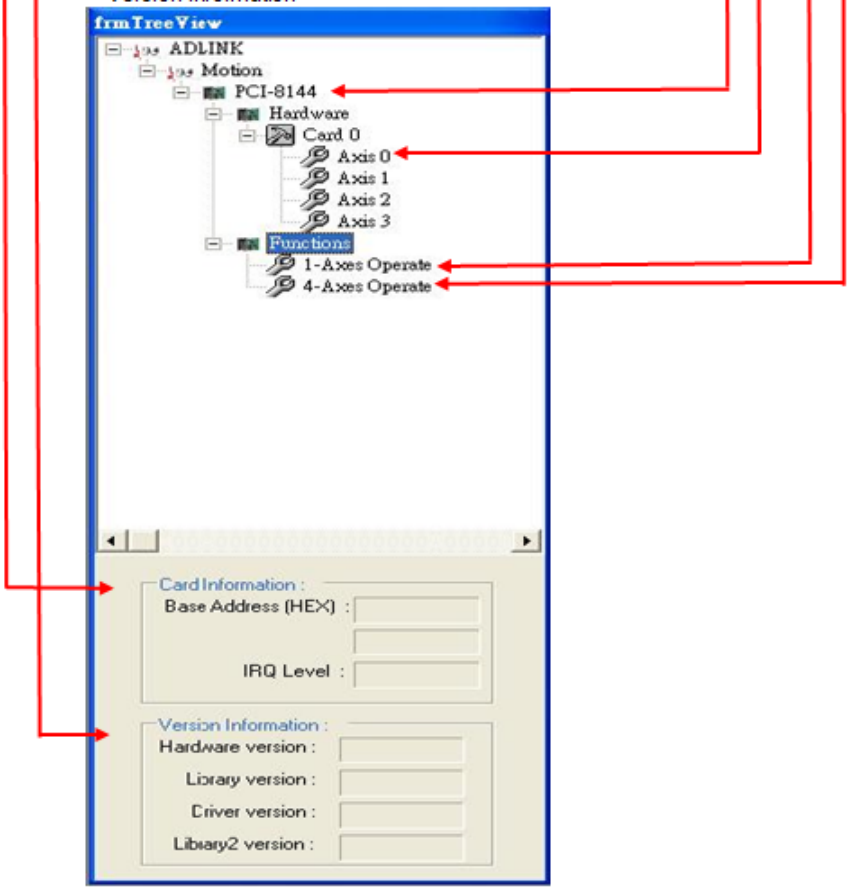
The main menu appears after running MotionCreatorPro. It is used to:



5.3.2 Select Menu

The select menu appears after running MotionCreatorPro. It is used to:

- Select operating card and axis
- Open Card Information Menu (refer to section 5.3.3)
- Open Configuration Menu (refer to section 5.3.4)
- Open Single Axis Operation Menu (refer to section 5.3.5)
- Open Four-Axis Operation Menu (refer to section 5.3.6)
- Show card information. Related function are : _8144_get_version(),
- Version Information



5.3.3 Card Information Menu

This menu shows Information about this card.

5.3.4 Configuration Menu

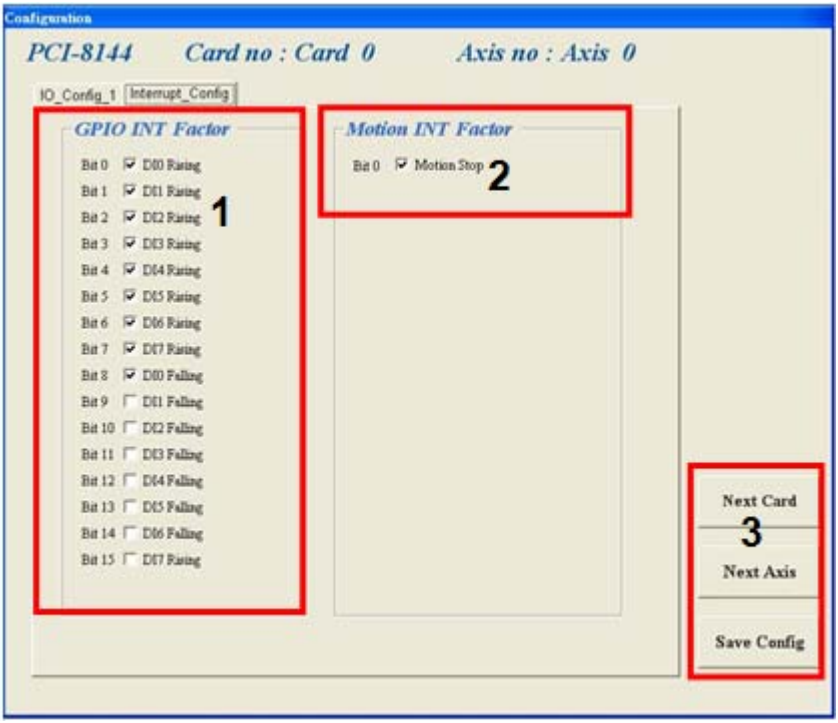
In this menu, users can configure EL, M_IO_Sensitivity, PulseLogic, SD_Singal and ORG_Stop.

1. **EL Logic**: Select the logic of the EL signal. The related function call is **`_8144_set_limit_logic()`**.
2. **M_IO_Sensitivity**: Select the configurations of the IO Sensitivity. The related function call is **`_8144_set_mio_sensitivity()`**.
3. **Pulse Logic**: Select the logic of the pulse mode. The related function call is **`_8144_set_pls_outmode()`**.
4. **SD_Singal**: Select the configuration of the SD singal. The related function call is **`_8144_enable_sd_signal()`**.
5. **ORG_Stop**: Select the configurations of the ORG_Stop.

The related function call is **`_8144_enable_org_stop()`**.

6. **Buttons**:
   - ▷ **Next Card**: Change operating card.
   - ▷ **Next Axis**: Change operating axis.
   - ▷ **Save Config**: Save current configuration to **`8144.ini`** and **`8144MC.ini`**.

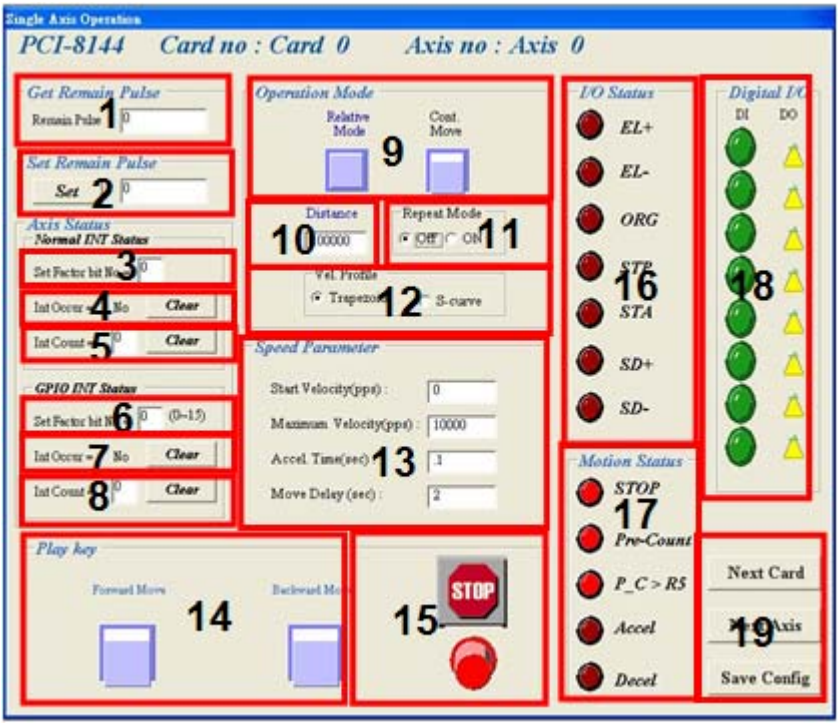
In this menu, users can configure motion INT factor and gpio INT factor.



1. **GPIO INT Factor**: Select factors to initiate the gpio event interrupt. The related function call is _8144_set_gpio_interrupt_factor().
2. **Motion INT Factor**: Select factors to initiate the event interrupt. The related function call is _8144_set_motion_interrupt_factor().
3. **Buttons**:
   - ▷ **Next Card**: Change operating card.
   - ▷ **Next Axis**: Change operating axis.
   - ▷ **Save Config**: Save current configuration to **`8144.ini`** and **`8144MC.ini`**.

5.3.5 Single Axis Operation Menu

In this menu, users can change the settings a selected axis, including velocity mode motion, preset relative motion.



1. **Get Remain Pulse**: Display the value of the remaining pulse. The related function is **`_8144_get_remaining_pulse()`**
2. **Set Remain Pulse**: Set the value of the remaining pulse. The related function is **`_8144_set_remaining_pulse()`**
3. **Set Factor bit No**:Set int_factor bit for normal motion interrupt. The related function call is **`_8144_set_motion_interrupt_factor()`**
4. **Int Occur**: Display if the interrupt happened. Show “Yes” If happened. Otherwise, show “No”.

5. **Int Count**: The counter would plus 1 if interrupt happened.
6. **Set Factor bit No**: Set int_factor bit for GPIO interrupt. The related function call is **`_8144_set_gpio_interrupt_factor()`**.
7. **Int Occur**: Display if the interrupt happened. Show “Yes” If happened. Otherwise, show “No”.
8. **Int Count**: The counter plus 1 if interrupt happened.
9. **Operation Mode**: Select operation mode.
   - ▷ **Relative Mode**: “Distance” will be used as relative displacement for motion. The related function is **`_8144_start_tr_move()`**, **`_8144_start_sr_move()`**.
   - ▷ **Cont. Move**: Velocity motion mode. The related function is **`_8144_tv_move()`**, **`_8144_start_sv_move()`**.
10. **Distance**: Set the relative distance for “Relative Mode.” It is only effective when “Relative Mode” is selected.
11. **Repeat Mode**: When “On” is selected, the motion will become repeat mode (forward<-->backward). It is only effective when “Relative Mode” is selected.
12. **Vel. Profile**: Select the velocity profile. Both Trapezoidal and S-Curve are available for “Relative Mode,” and “Cont. Move.”
13. **Motion Parameters**: Set the parameters for single axis motion.
    - ▷ **Start Velocity**: Set the start velocity of motion in units of PPS. In “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**: Set the maximum velocity of motion in units of PPS. In “Relative Mode,” only the value is effective. For example, -5000.0 is the same as 5000.0.

In “Cont. Move,” both the value and sing is effective. –5000.0 means 5000.0 in the minus direction.

- ▷ **Accel. Time**: Set the acceleration time in units of second. Tdec is the same as tacc.
- ▷ **Move Delay**: This setting is effective only when repeat mode is set “On.” It will cause the 8144 to delay for a specified time before it continues to the next motion.

14. **Play Key**:

- ▶ **Left play button**: Clicking this button will cause the 8144 start to outlet pulses according to previous setting.
  - ▷ In “Relative Mode,” it causes the axis to move forward.
  - ▷ In “Cont. Move,” it causes the axis to start to move according to the velocity setting.
- ▶ **Right play button**: Clicking this button will cause the 8144 start to outlet pulses according to previous setting.
  - ▷ In “Relative Mode,” it causes the axis to move backwards.
  - ▷ In “Cont. Move,” it causes the axis to start to move according to the velocity setting, but in the opposite direction.

15. **Stop Button**: Clicking this button will cause the 8144 to stop. The related function is **`_8144_emg_stop()`**.

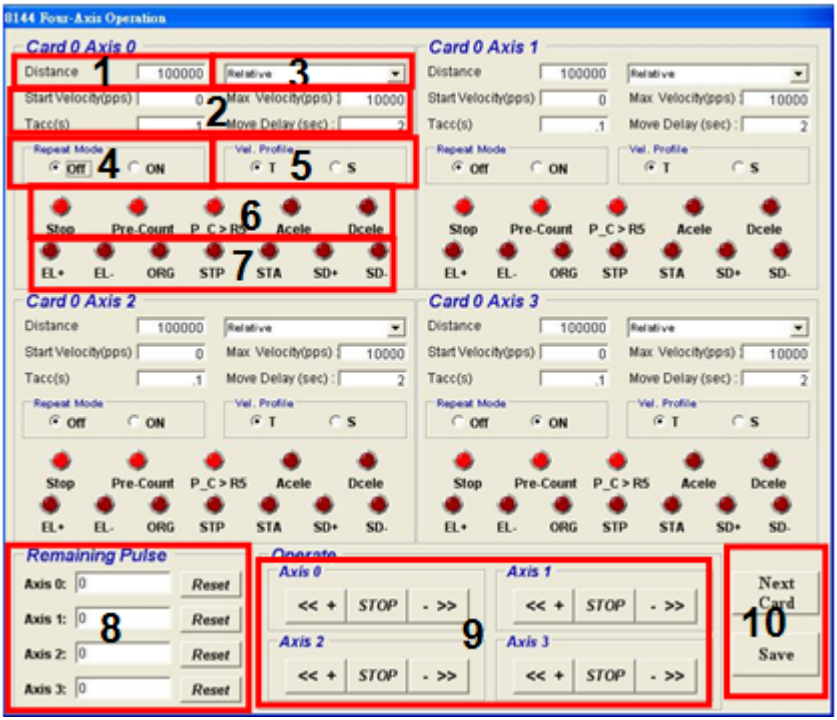
16. **I/O Status**: The status of motion I/O. Light-On means Active, while Light-Off indicates inactive. The related function is **`_8144_get_mio_status()`**.

17. **Motion Status**: The status of motion when running. Light-On means Active, while Light-Off indicates inactive. The related function is **`_8144_motion_status()`**.

18. **Digital I/O**: Display and set Digital I/O. The related function is:
    `_8144_get_gpio_output()`,
    `_8144_get_gpio_input()`,
    `_8144_set_gpio_output()`.
19. **Buttons**:
    - ▷ **Next Card**: Change operating card.
    - ▷ **Next Axis**: Change operating axis.
    - ▷ **Save Config**: Save current configuration to **`8144.ini`** and **`8144MC.ini`**.

5.3.6 Four-Axis Operation Menu

In this menu, users can change the settings four selected axis, including velocity mode motion, preset relative motion.



1. **Distance**: Set the relative distance for “Relative Mode.” It is only effective when “Relative Mode” is selected.
2. **Motion Parameters**: Set the parameters for single axis motion.
   - ▷ **Start Velocity**: Set the start velocity of motion in units of PPS. In “Relative Mode”, only the value is effective. For example, -100.0 is the same as 100.0.
   - ▷ **Maximum Velocity**: Set the maximum velocity of motion in units of PPS. In “Relative Mode”, only the value is effective. For example, -5000.0 is the same as 5000.0.

- ▷ **Tacc**: Set the acceleration time in units of second and tdec is the same as tacc.

3. **Operation Mode**: Select operation mode.
   - ▷ **Relative Mode**: “Distance” will be used as relative displacement for motion. The related function is **`_8144_start_tr_move()`**, **`_8144_start_sr_move()`**.
4. **Repeat Mode**: When “On” is selected, the motion will become repeat mode (forward<-->backward). It is only effective when “Relative Mode” is selected
5. **Vel. Profile**: Select the velocity profile. Both Trapezoidal and S-Curve are available for Relative Mode.
6. **Motion Status**: The status of motion. Light-On means Active, while Light-Off indicates inactive. The related function is **`_8144_motion_status()`**.
7. **I/O Status**: The status of motion I/O. Light-On means Active, while Light-Off indicates inactive. The related function is **`_8144_get_mio_status()`**.
8. Remaining Pulse: Display the value of the remaining pulse. The related function is **`_8144_get_remaining_pulse()`**
9. **Play Key**:

- ▶ **Left play button**: Clicking this button will cause the 8144 start to outlet pulses according to previous setting.
  - ▷ In “Relative Mode,” it causes the axis to move forward.
- ▶ **Right play button**: Clicking this button will cause the 8144 start to outlet pulses according to previous setting.
  - ▷ In “Relative Mode,” it causes the axis to move backwards.
- ▶ **Stop Button**: Clicking this button will cause the 8144 to stop. The related function is _8144_emg_stop().

10. **Buttons**:
    - ▷ **Next Card**: Change operating card.
    - ▷ **Save Config**: Save current configuration to 8144.ini and 8144MC.ini.

5.3.7 Help Menu

In this menu, users can Click Mouse Right Key to show Help Information.

6 Function Library

This chapter describes the supporting software for the 8144 card. User can use these functions to develop programs in C, C++, or Visual Basic. If Delphi is used as the programming environment, it is necessary to transform the header files, 8144.h manually.

6.1 List of Functions

6.2 C/C++ Programming Library

This section details all the functions. The function prototypes and some common data types are declared in pci_8144.h. We suggest 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

Table 6-1: Data type definitions

The functions of the PCI-8144 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. **`_8144_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_8144_initial()`**.

6.3 System and Initialization

_8144_initial

Device initialization

Description:

This function is used to initialize PCI-8144 cards and assign hardware resources. All 8144 cards must be initialized by this function before calling other functions in your applications. By setting the parameter “ManualId”, user can choose the type that the card’s ID is assigned manually or automatically. In the end of your application, you should use the function, _8144_close(), to release its resources.

Syntax:

```
I16 _8144_initial( I32 *CardIdInBit, I16 ManualId
      );
B_8144_initial( CardIdInBit As Long, ByVal
      ManualId As Integer) As Integer
```

Parameters:

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

Note: if the value of CardIdInBit is 0x3, that means there are 2 cards in your system and those card ID are 0 and 1 respectively.

`I16 ManualId`: Enable the On board dip switch (SW1) to decide theCard ID. [0:By system assigned, 1:By dip switch]

Return Values:

```
ERR_NoError
ERR_OpenDriverFailed
ERR_InsufficientMemory
ERR_CardIdDuplicate
ERR_NoDeviceFound
```

Example:

```
I16 ret; // return value
I32 CardIdInBit;
```

```
I16 ManualId = 0; //By system assigned
ret = _8144_initial( &CardIdInBit, ManualId );
…// Do something
ret = _8144_close(); //Close all PCI-8144 cards
      in the system
```

See also:

```
_8144_close();
```

_8144_close

Devices close

Description:

This function is used to close all PCI-8144 cards in the system and release its' resources, which must be called at the end of your applications.

Syntax:

```
I16 _8144_close()
B_8144_close() As Integer
```

Parameters:

Return Values:

```
ERR_NoError : Success.
```

Example:

```
I16 ret; // return value
I32 CardIdInBit;
I16 ManualId = 0; //By system assigned
ret = _8144_initial( &CardIdInBit, ManualId );
…// Your applications
ret = _8144_close(); //Close all PCI-8144 cards
     in the system
```

See also:

```
_8144_initial();
```

_8144_get_version

Get software and hardware version numbers

Description:

The user can get software and hardware versions by this function.

Syntax:

```
I16 _8144_get_version( I16 CardId, I32 *Dll_1,
     I32 *Dll_2, I32 *FirmwareVersion, I32
     *DriverVersion );
B_8144_get_version( ByVal CardId As Integer,
     Dll_1 As Long, Dll_2 As Long,
     FirmwareVersion As Long, DriverVersion As
     Long ) As Integer
```

Parameters:

`I16 CardId`: The card ID number

`I32 *Dll_1`: The 8144.DLL version information

`I32 *Dll_2`: The MC4541.DLL version information

`I32 *FirmwareVersion`: The information of 8144 card firmware version

`I32 *DriverVersion`: The information of WDM driver.

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_Win32Error
ERR_NoError
```

Example:

```
I16 CardId = 0;
I32 DLL_1, DLL_2, FirmwareVersion, DriverVersion;
I16 ret = _8144_get_version(CardId, &Dll_1,
     &Dll_2, &FirmwareVersion, &DriverVersion );
…
```

See also:

_8144_set_security_key

Set the security password

Description:

This function is used to set a security code to the PCI card in EEPROM. The security code will never dispear even if the system is powered off.

Syntax:

```
I16 _8144_set_security_key(I16 CardId, U16
     OldPassword, U16 NewPassword)
B_8144_set_security_key(ByVal CardId As Integer,
     ByVal OldPassword As Integer, ByVal
     NewPassword As Integer)As Integer
```

Parameters:

`I16 CardId`: The card ID number.

`U16 OldPassword`: The current password stored in card.

`U16 NewPassword`: New password.

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_GetEEPROM
ERR_SetEEPROM
ERR_NoError
```

Example:

```
I16 ret; //return code
I16 CardId = 0; // card ID
U16 OldPassword = 0; // current password stored
     in card
U16 NewPassword = 0x1234; //new password
If(_8144_reset_security_key(CardId)) //reset the
     password to default value (0)
{// return error}
if(_8144_set_security_key( CardId, OldPassword,
     NewPassword)) //set a new password
{//return error}
```

```
ret = _8144_check_security_key(CardId,
     NewPassword); //verify the new password
if( ret == ERR_NoError )
  // security pass
else
  // security failed
```

See also:

_8144_check_security_key
_8144_reset_security_key

_8144_check_security_key

Varily the security password

Description:

This funciton is used to verify the security code which the user set by the function “_8144_set_security_key”.

Syntax:

```
I16 _8144_check_security_key(I16 CardId, U16
     Password)
B_8144_check_security_key(ByVal CardId As
     Integer, ByVal Password As Integer) As
     Integer
```

Parameters:

`I16 CardId`: The card ID number.

`U16 Password`: The password number

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_GetEEPROM
ERR_NoError
```

Example:

```
I16 ret; //return code
I16 CardId = 0; // card ID
U16 OldPassword = 0; // current password stored
     in card
U16 NewPassword = 0x1234; //new password
If(_8144_reset_security_key(CardId)) //reset the
     password to default value (0)
{// return error}
if(_8144_set_security_key( CardId, OldPassword,
     NewPassword)) //set a new password
{//return error}
ret = _8144_check_security_key(CardId,
     NewPassword); //verify the new password
if( ret == ERR_NoError )
  // security pass
```

```
else
  // security failed
```

See also:

```
_8144_set_security_key
_8144_reset_security_key
```

_8144_reset_security_key

Reset the security password to default value

Description:

By this function, Users can reset the security code which is stored on the PCI card to default value. The default security code is0.

Syntax:

```
I16 _8144_reset_security_key(I16 CardId)
B_8144_reset_security_key(ByVal CardId As
     Integer) As Integer
```

Parameters:

`I16 CardId`: The card ID number

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_SetEEPROM
ERR_NoError
```

Example:

```
I16 ret; //return code
I16 CardId = 0; // card ID
U16 OldPassword = 0; // current password stored
     in card
U16 NewPassword = 0x1234; //new password
If(_8144_reset_security_key(CardId)) //reset the
     password to default value (0)
{// return error}
if(_8144_set_security_key( CardId, OldPassword,
     NewPassword)) //set a new password
{//return error}
ret = _8144_check_security_key(CardId,
     NewPassword); //verify the new password
if( ret == ERR_NoError )
  // security pass
else
  // security failed
```

See also:

```
_8144_set_security_key
_8144_check_security_key
```

_8144_config_from_file

Configure settings from file

Description:

This function is used to load the configeration of the PCI-8144 according to specified file. By using Motion Creater Rro, user could test and configure the 8144 correctly. After saving the configuration, the file would be existed in user's system directory as 8144.ini.

When this function is executed, all 8144 cards in the system will be configured as the following functions were called according to parameters recorded in 8144.ini.

```
_8144_set_limit_logic
_8144_set_mio_sensitivity
_8144_set_pls_outmode,
_8144_enable_sd_signal
```

Syntax:

```
I16 _8144_config_from_file()
B_8144_config_from_file()As Integer
```

Parameters:

Return Values:

```
ERR_NoError
```

Example:

See also:

```
_8144_set_limit_logic
_8144_set_mio_sensitivity
_8144_set_pls_outmode
_8144_enable_sd_signal
```

6.4 Motion Interface I/O

_8144_set_limit_logic

Set the logic of PEL/MEL input signals

Description:

Set the active logic of specified axis's PEL/MEL input signal.

Syntax:

```
I16 _8144_set_limit_logic(I16 AxisNo, I16
      LimitLogic )
B_8144_set_limit_logic(ByVal AxisNo As Integer,
      ByVal LimitLogic As Integer ) As Integer
```

Parameters:

`I16 AxisNo`: Axis number of Target Axis.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

`I16 LimitLogic`: The logic of PEL/MEL input signal

- ▷ 0: Negative logic
- ▷ 1: Positive logic

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_NoError
```

Example:

```
I16 ret; // return value
I16 AxisNo = 0; //Axis number
I16 LimitLogic = 1; //Positive logic
ret = _8144_set_limit_logic(AxisNo, LimitLogic);
     //EL logic setting
```

See also:

```
_8144_get_limit_logic
_8144_get_mio_status
_8144_set_mio_sensitivity
```

_8144_get_limit_logic

Get the logic of PEL/MEL input signals

Description:

Get the information of PEL/MEL input signal logic setting.

Syntax:

```
I16 _8144_get_limit_logic(I16 AxisNo, I16
      *LimitLogic )
B_8144_get_limit_logic(ByVal AxisNo As Integer,
      LimitLogic As Integer )
```

Parameters:

`I16 AxisNo`: Axis number of Target Axis.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

`I16 *LimitLogic`: Theinformation of logic of PEL/MEL input signal.

- ▷ 0: Negative logic
- ▷ 1: Positive logic

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_NoError
```

Example:

```
I16 ret; //return code
I16 AxisNo = 0;
I16 LimitLogic;
ret = _8144_get_limit_logic(I16 AxisNo,
     &LimitLogic );
```

See also:

```
_8144_set_limit_logic
_8144_get_mio_status
_8144_set_mio_sensitivity
```

_8144_get_mio_status

Get the status of motion I/O signals

Description:

Get all of motion I/O, status of each axis

Syntax:

```
I16 _8144_get_mio_status(I16 AxisNo, I16
     *MotionIoStatusInBit )
B_8144_get_mio_status(ByVal AxisNo As Integer,
     MotionIoStatusInBit As Integer ) As Integer
```

Parameters:

`I16 AxisNo`: Axis number of Target Axis.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

`I16 *MotionIoStatusInBit`: Statuses of Motion I/O. The definition of each bit as follow

Bit	Name	Description
0	MEL	Negative Limit Switch
1	PEL	Positive Limit Switch
2	ORG	Origin Switch
3	STP	STP pin status(EMG)
4	STA	STA pin status
5	MSD	Negative Slow Down signal input

Bit	Name	Description
6	PSD	Positive Slow Down signal input
7~	-	Reserve

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_NoError
```

Example:

```
I16 ret;
I16 AxisNo = 0; //axis number
I16 MotionIoStatusInBit; //The information of
     motion I/O in bit
ret = _8144_get_mio_status(AxisNo,
     &MotionIoStatusInBit );
…
```

See also:

```
_8144_get_limit_logic
_8144_set_ limit_logic
_8144_set_mio_sensitivity
```

_8144_set_mio_sensitivity

Set the sensitive of motion I/O signals

Description:

By this function, to set “low sensitivity” to reduce the sensitivity to signals on the ORG, P/MEL and STP signal. Pulse signals shorter than 4 reference clock cycles(approx. 800ns) will be ignored.

Syntax:

```
I16 _8144_set_mio_sensitivity( I16 AxisNo, I16
     HighOrLow )
B_8144_set_mio_sensitivity( ByVal AxisNo As
     Integer, ByVal HighOrLow As Integer ) As
     Integer
```

Parameters:

`I16 AxisNo`: Axis number of Target Axis.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

`I16 HighOrLow`: Sensitivity setting

- ▷ 0: HIGH_SENSITIVITY
- ▷ 1: LOW_SENSITIVITY

Return Values:

`ERR_ParametersInvalid`

```
ERR_DeviceNotInitial
ERR_NoError
```

Example:

```
I16 ret; //return value
I16 AxisNo = 0; //axis number
I16 HighOrLow = 1; //set to low sensitivity
ret = _8144_set_mio_sensitivity(AxisNo, HighOrLow
      );
…
```

See also:

```
_8144_get_limit_logic
_8144_set_ limit_logic
_8144_get_mio_status
```

_8144_set_pls_outmode

Set the logic of pulse output signal

Description:

By this function, you can change the logic of pulse output signal.

Syntax:

```
I16 _8144_set_pls_outmode( I16 AxisNo, I16
      PulseLogic )
B_8144_set_pls_outmode( ByVal AxisNo As Integer,
      ByVal PulseLogic As Integer ) As Integer
```

Parameters:

`I16 AxisNo`: Axis number of Target Axis.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

`I16 PulseLogic`: The setting of logic of pulse output signal

- ▷ 0: Negative logic
- ▷ 1: Positive logic

Direction	0: Negative logic		1: Positive logic	
	+PO	-PO	+PO	-PO
(+)		H		L
(-)	H		L	

(*) H: High, L: Low

Return Values:

```
ERR_AxisNumber
ERR_DeviceNotInitial
ERR_ParametersInvalid
ERR_NoError
```

Example:

```
I16 AxisNo = 0; //Axis 0
I16 PulseLogic = 1; //Positive logic
I16 ret = _8144_set_pls_outmode(AxisNo,
     PulseLogic ); //Set command to the card.
…
```

See also:

_8144_set_pls_outmode2

Description:

This function is used to set the output pulse mode and its logic.

Syntax:

```
I16 _8144_set_pls_outmode2 ( I16 AxisNo, I16
     PulseMode, I16 PulseLogic );
B_8144_enable_get_command( ByVal AxisNo As
     Integer, ByVal PulseMode As Integer ByVal
     PulseLogic As Integer ) As Integer
```

Parameters:

`I16 AxisNo`: Axis number of Target Axis.

`I16 PulseMode`: Output pulse mode.

0: CW/CCW

1: OUT/DIR

`I16 PulseLogic`: Pulse output logic.

0: Negative logic

1: Positive logic

Return Values:

I16 Error code. Refer to error code table.

Example:

```
I16 ret; //Return code
I16 PulseMode = 0; //CW/CCW
I16 PulseLogic = 0; //Negative logic
ret = _8144_set_pls_outmode2( AxisNo, PulseMode,
     PulseLogic );
…
```

See also:

`_8144_set_pls_outmode()`

6.5 Motion

_8144_tv_move

Accelerate an axis to a constant velocity with trapezoidal profile

Description:

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

This function has warning return code. It means that the speed profile that users specified doesn't make sense or out of ASIC can do. It will adjust the parameter automatically and return a warning code, and the motion card still output pulses to control the axis.

Syntax:

```
I16 _8144_tv_move( I16 AxisNo, F64 StrVel, F64
     MaxVel, F64 Tacc )
B_8144_tv_move( ByVal AxisNo As Integer, ByVal
     StrVel As Double, ByVal MaxVel As Double,
     F64 Tacc As Double) As Integer
```

Parameters:

`I16 AxisNo`: The axis number is designated to receive the command.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3
1	0	4
	1	5
	2	6
	3	7

Card ID	Physical axis	AxisNo
2	0	8
	...	...

F64 StrVel: Starting velocity (pulse/sec)

F64 MaxVel: Maximum velocity (pulse/sec)

F64 Tacc: Specified acceleration and deceleration time (sec)

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
WAR_AccDecTimeToSmall
WAR_AccDecTimeToLarge
ERR_NoError
```

Example:

```
I16 ret; //return code
I16 AxisNo = 0; // axis number
F64 StrVel = 500; // starting velocity
F64 MaxVel = 50000; // maximum velocity
F64 Tacc = 0.5; // acceleration/deceleration time
     (sec)
ret = _8144_tv_move( AxisNo, StrVel, MaxVel, Tacc
     );
```

See also:

```
_8144_sv_move
_8144_emg_stop
_8144_dec_stop
_8144_slow_down
_8144_enable_org_stop
_8144_enable_sd_signal
_8144_set_external_start
```

_8144_sv_move

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

Description:

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

This function has warning return code. It means that the speed profile that users specified doesn't make sense or out of ASIC can do. It will adjust the parameter automatically and return a warning code, and the motion card still output pulses to control the axis.

Syntax:

```
I16 _8144_sv_move( I16 AxisNo, F64 StrVel, F64
     MaxVel, F64 Tacc )
B_8144_sv_move( ByVal AxisNo As Integer, ByVal
     StrVel As Double, ByVal MaxVel As Double,
     F64 Tacc As Double) As Integer
```

Parameters:

`I16 AxisNo`: The axis number is designated to receive the command.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

F64 StrVel: Starting velocity (pulse/sec)

F64 MaxVel: Maximum velocity (pulse/sec)

F64 Tacc: Specified acceleration and deceleration time (sec)

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
WAR_AccDecTimeToSmall
WAR_AccDecTimeToLarge
ERR_NoError
```

Example:

```
I16 ret; //return code
I16 AxisNo = 0; // axis number
F64 StrVel = 500; // starting velocity
F64 MaxVel = 50000; // maximum velocity
F64 Tacc = 0.5; // acceleration/deceleration time
     (sec)
ret = _8144_sv_move( AxisNo, StrVel, MaxVel, Tacc
     );
```

See also:

```
_8144_tv_move
_8144_emg_stop
_8144_dec_stop
_8144_slow_down
_8144_enable_org_stop
_8144_enable_sd_signal
_8144_set_external_start
_8144_get_remaining_pulse
_8144_set_remaining_pulse
_8144_get_tv_move_profile
```

_8144_start_tr_move

Begin a relative trapezoidal profile move

Description:

This function causes the axis to accelerate form a starting velocity (StrVel), rotate at constant velocity (MaxVel), and decelerate to stop at the relative distance with symmetrical trapezoidal profile. The acceleration and deceleration time is specified by a parameter “Tacc”. The moving direction is determined by the sign of the Distance parameter. It does not let the program wait for motion completion but immediately returns control to the program.

This function has warning return code. It means that the speed profile that users specified doesn’t make sense or out of ASIC can do. It will adjust the parameter automatically and return a warning code, and the motion card still output pulses to control the axis.

Syntax:

```
I16 _8144_start_tr_move( I16 AxisNo, F64
     Distance, F64 StrVel, F64 MaxVel, F64 Tacc
     );
B_8144_start_tr_move( ByVal AxisNo As Integer,
     ByVal Distance As Double, ByVal StrVel As
     Double, ByVal MaxVel As Double, ByVal Tacc
     As Double ) As Integer
```

Parameters:

`I16 AxisNo`: The axis number is designated to receive the command.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3

Card ID	Physical axis	AxisNo
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

F64 Distance: Specified relative distance to move (pulse)

F64 StrVel: Starting velocity (pulse/sec)

F64 MaxVel: Maximum velocity (pulse/sec)

F64 Tacc: Specified acceleration and deceleration time (sec)

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
WAR_AccDecTimeToSmall
WAR_AccDecTimeToLarge
WAR_RampDownPointExceed
ERR_NoError
```

Example:

```
I16 ret;
I16 AxisNo = 0; // axis number
F64 Distance = 50000.0; //reletive distance
F64 StrVel = 500.0; // starting velocity
F64 MaxVel = 10000.0; // maximum velocity
F64 Tacc = 0.5; // acceleration/deceleration time
     (sec)
ret = _8144_start_tr_move( AxisNo, Distance,
     StrVel, MaxVel, Tacc );
```

See also:

```
_8144_start_sr_move
_8144_set_external_start
_8144_emg_stop
_8144_dec_stop
_8144_slow_down
_8144_enable_org_stop
```

```
_8144_enable_sd_signal
_8144_get_remaining_pulse
_8144_set_remaining_pulse
_8144_get_sv_move_profile
```

_8144_start_sr_move

Begin a relative S-curve profile move

Description:

This function causes the axis to accelerate from a starting velocity (StrVel), rotate at constant velocity (MaxVel), and decelerates to stop at the relative distance with S-curve profile. The acceleration and deceleration time is specified by a parameter “Tacc”. The moving direction is determined by the sign of the Distance parameter. This command does not let the program wait for motion completion, but immediately returns control to the program.

This function has warning return code. It means that the speed profile that users specified doesn’t make sense or out of ASIC can do. It will adjust the parameter automatically and return a warning code, and the motion card still output pulses to control the axis.

Syntax:

```
I16 _8144_start_sr_move( I16 AxisNo, F64
     Distance, F64 StrVel, F64 MaxVel, F64 Tacc
     );
B_8144_start_sr_move( ByVal AxisNo As Integer,
     ByVal Distance As Double, ByVal StrVel As
     Double, ByVal MaxVel As Double, ByVal Tacc
     As Double ) As Integer
```

Parameters:

`I16 AxisNo`: The axis number is designated to receive the command.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3

Card ID	Physical axis	AxisNo
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

F64 Distance: Specified relative distance to move (pulse)

F64 StrVel: Starting velocity (pulse/sec)

F64 MaxVel: Maximum velocity (pulse/sec)

F64 Tacc: Specified acceleration and deceleration time (sec)

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
WAR_AccDecTimeToSmall
WAR_AccDecTimeToLarge
WAR_RampDownPointExceed
ERR_NoError
```

Example:

```
I16 ret;
I16 AxisNo = 0; // axis number
F64 Distance = 50000.0; //reletive distance
F64 StrVel = 500.0; // starting velocity
F64 MaxVel = 10000.0; // maximum velocity
F64 Tacc = 0.5; // acceleration/deceleration time
     (sec)
ret = _8144_start_sr_move( AxisNo, Distance,
     StrVel, MaxVel, Tacc );
```

See also:

```
_8144_start_tr_move
_8144_set_external_start
_8144_emg_stop
_8144_dec_stop
_8144_slow_down
_8144_enable_org_stop
```

```
_8144_enable_sd_signal
_8144_get_remaining_pulse
_8144_set_remaining_pulse
```

_8144_enable_get_command

Description:

This function is used to enable “get_command()” function. If users enable “get_command” function, users can use **_8144_get_command**() function to get current command position.

When the axis is in motion, you can not issue this function. **When “get command” is enabled, the command position will be reset to zero and you can not change speed of axes on the fly**.

Syntax:

```
I16 _8144_enable_get_command( I16 AxisNo, I16
     Enable );
B_8144_enable_get_command( ByVal AxisNo As
     Integer, ByVal Enable As Integer ) As
     Integer
```

Parameters:

I16 AxisNo: Axis number of Target Axis.

I16 Enable:

0: Disable get command position function.

1: Enable get command position function.

Return Values:

```
I16 Error code. Refer to error code table.
```

Example:

```
I16 ret;
ret = _8144_enable_get_command( AxisNo, 1 ); //
     Enable get command function.
…
```

See also:

```
_8144_get_command
_8144_set_command
```

_8144_get_command

Description:

This function is used to get command position of an axis. The command position is calculated from hardware preset counter (Get by **_8144_enable_get_command**). Users must enable this function before use it by the function: "**_8144_enable_get_command**".

Syntax:

```
I16 _8144_get_command( I16 AxisNo, I32 *Cmd );
B_8144_get_command( ByVal AxisNo As Integer, Cmd
     As Long ) As Integer
```

Parameters:

I16 AxisNo: Axis number of Target Axis.

I32 *Cmd: Retrieve command position.

Return Values:

```
I16 Error code. Refer to error code table.
```

Example:

```
I16 ret;
I32 cmd = 0;
ret = _8144_get_command( AxisNo, &cmd );
if( ret == ERR_NoError )
{
     //…do something
}
```

See also:

```
_8144_enable_get_command
_8144_set_command
_8144_get_remaining_pulse
```

_8144_set_command

Description:

This function is used to set command position of an axis when “get command” function has been enabled.

Syntax:

```
I16 _8144_set_command( I16 AxisNo, I32 Cmd );
B_8144_set_command( ByVal AxisNo As Integer,
     ByVal Cmd As Long ) As Integer
```

Parameters:

`I16 AxisNo`: Axis number of Target Axis.

`I32 Cmd`: set command position.

Return Values:

```
I16 Error code. Refer to error code table.
```

Example:

```
I16 ret;
I32 cmd = 0;
ret = _8144_set_command( AxisNo, cmd );
if( ret == ERR_NoError )
{
     //…
}
```

See also:

```
_8144_enable_get_command
_8144_get_command
```

_8144_set_external_start

Set inhibit start by STA input signal

Description:

By this function, the operation start can be inhibited. When a motion command is issued, the axis will remain stopped. Then, when the STA input signal is active, the inhibit is released, and the axis will start move. Disable this function, the axis will start immediately. Notice that, all axes use only one STA input signal.

Syntax:

```
I16 _8144_set_external_start( I16 AxisNo, I16
      Enable )
B_8144_set_external_start( ByVal AxisNo As
      Integer, ByVal Enable As Integer )As Integer
```

Parameters:

`I16 AxisNo`: The axis number is designated to receive the command.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

`I16 Enable`:

- ▷ 0: disable (start move immediately)
- ▷ 1: enable(start move by STA input signal)

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_NoError
```

Example:

```
I16 AxisNo = 0;
I16 Enable = 1; //Enable STA start
_8144_set_external_start(AxisNo, Enable ); //
     Enable STA start
_8144_start_sr_move( AxisNo, Distance, StrVel,
     MaxVel, Tacc );
// motion will not start until STA signal active…
Enable = 0; //Disable STA start
_8144_set_external_start(AxisNo, Enable ); //
     Dsiable STA start
```

See also:

```
_8144_tv_move
_8144_sv_move
_8144_start_tr_move
_8144_start_sr_move
```

_8144_emg_stop

Immediately stop

Description:

This function is used to immediately stop an axis.

Syntax:

```
I16 _8144_emg_stop( I16 AxisNo )
B_8144_emg_stop( ByVal AxisNo As Integer)As
     Integer
```

Parameters:

I16 AxisNo: The axis number is designated to receive the command.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_NoError
```

Example:

```
ret = _8144_tv_move( AxisNo, StrVel, MaxVel, Tacc
     ); //perform a tv move.
…
ret = _8144_emg_stop( AxisNo ); //immediately
     stop
```

See also:

`_8144_dec_stop`

_8144_dec_stop

Decelerate to stop

Description:

This function is used to decelerate an axis to stop with a trapezoidal or S-curve profile. This function is also useful when a preset move (start_tr_move, start_sr_move) is performed. Note: The velocity profile is decided by original motion profile.

Syntax:

```
I16 _8144_dec_stop( I16 AxisNo )
B_8144_dec_stop( ByVal AxisNo As Integer )As
     Integer
```

Parameters:

`I16 AxisNo`: The axis number is designated to receive the command.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_NoError
```

Example:

```
ret = _8144_tv_move( AxisNo, StrVel, MaxVel, Tacc
     ); //perform a tv move.
```

```
…
ret = _8144_dec_stop( AxisNo ); //slow down to
      the start velocity then stop
```

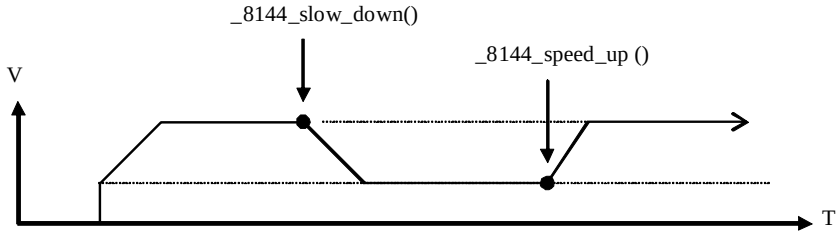
See also:

_8144_emg_stop

_8144_speed_up

Description:

This function is used to re-accelerate an axis to maximum velocity with a trapezoidal or S-curve profile. This function is also useful when a preset move (start_tr_move, start_sr_move) is performed. Note: The velocity profile is decided by original motion profile.



Syntax:

```
I16 _8144_speed_up( I16 AxisNo )
B_8144_speed_up( ByVal AxisNo As Integer ) As
     Integer
```

Parameters:

`I16 AxisNo`: Axis number of Target Axis.

Return Values:

```
I16 Error code. Refer to error code table.
```

Example:

```
I16 ret; //Return code
// perform a move function. Eg. _8144_tv_move()
_8144_slow_down( AxisNumber ); //Deceleration to
     start velocity.
//…
ret = _8144_speed_up( AxisNumber ); //Re-
     acceleration to max. velocity
…
```

See also:

```
_8144_slow_down()
```

_8144_slow_down

Slow down to start velocity

Description:

This function is used to decelerate an axis with a trapezoidal or S-curve profile. This function is also useful when a preset move (start_tr_move, start_sr_move) is performed. Note: The velocity profile is decided by original motion profile.

Syntax:

```
I16 _8144_slow_down( I16 AxisNo )
B_8144_slow_down( ByVal AxisNo As Integer)As
     Integer
```

Parameters:

`I16 AxisNo`: The axis number is designated to receive the command.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_NoError
```

Example:

```
ret = _8144_tv_move( AxisNo, StrVel, MaxVel, Tacc
     ); //perform a tv move.
```

```
…
ret = _8144_ slow_down ( AxisNo ); //slow down to
      the start velocity
…
_8144_tv_move( AxisNo, StrVel, MaxVel, Tacc ); //
      acceleration to previous maximum velocity on
      the fly
```

See also:

_8144_enable_org_stop

Enable the stop when org signal is active

Description:

By this function, the user can enable/disable the ORG stop function. When this function is enable, the axis will stop instantly when ORG signal is turn ON. With those motion functions, user can perform a homing return move.

Regardless of whether or not ORG stop is enable or disable. The user can monitor the ORG input signal's status by "_8144_get_mio_status" function, and set the sensitivity of ORG input signal by "_8144_set_mio_sensitivity".

Syntax:

```
I16 _8144_enable_org_stop( I16 AxisNo, I16 Enable
      )
B_8144_enable_org_stop( ByVal AxisNo As Integer,
     ByVal Enable As Integer )As Integer
```

Parameters:

`I16 AxisNo`: The axis number is designated to receive the command.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

`I16 Enable`:

- ▷ 0: Disable: the axis will not stop when ORG signal is active
- ▷ 1: Enable: the axis will stop when ORG signal is active

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_NoError
```

Example:

```
I16 AxisNo = 0; // axis number
I16 Enable = 1; //Enable ORG stop
ret = _8144_enable_org_stop(AxisNo, Enable ); //
      set ORG stop enable
ret = _8144_sv_move( AxisNo, StrVel, MaxVel, Tacc
      ); //perform a sv_move
…//monitor ORG signal or check motion done or
      using interrupt function…
Enable = 0; //Disable ORG stop
ret = _8144_enable_org_stop(AxisNo, Enable ); //
      set ORG stop disable
```

See also:

```
_8144_get_mio_status
_8144_set_mio_sensitivity
_8144_wait_single_motion_interrupt
```

_8144_enable_sd_signal

Enable the slow down when SD signal is active

Description:

When this function is enable, and if an SD signal of the same polarity as the motor rotation is turn ON, the motor will start decelerating. If the SD signal goes OFF, the motor will accelerate again.

Regardless of whether or not this function is enable or disable. The user can monitor the SD input signal's status by "_8144_get_mio_status" function.

This is used to reduce mechanical shock when stopping in home move or when using EL signal.

Syntax:

```
I16 _8144_enable_sd_signal( I16 AxisNo, I16
     Enable )
B_8144_enable_sd_signal( ByVal AxisNo As Integer,
     ByVal Enable As Integer ) As Integer
```

Parameters:

`I16 AxisNo`: The axis number is designated to receive the command.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

`I16 Enable`:

- 0: Disable: the axis will not slow down when SD signal is active
- 1: Enable: the axis will slow down when SD signal is active

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_NoError
```

Example:

```
I16 ret; // return code
I16 AxisNo = 0; //axis number
I16 Enable = 1; // Enable slow down by SD signal
ret = _8144_enable_sd_signal(AxisNo, Enable ); //
      set command
…//move functions
```

See also:

```
_8144_get_mio_status
```

_8144_home_move

Description:

This function is used to perform a software based home return operation. The homing status can be checked by the "**_8144_home_status**" function.

Syntax:

```
I16 _8144_home_move( I16 AxisNo, F64 StrVel, F64
     MaxVel, F64 Tacc, I16 AccType );
B_8144_home_move( ByVal AxisNo As Integer, ByVal
     StrVel As Double,ByVal MaxVel As Double,
     ByVal Tacc As Double, ByVal AccType As
     Integer );
```

Parameters:

I16 AxisNo: Axis number of Target Axis.

F64 StrVel: Start homing velocity. Unit: pulse/second.

F64 MaxVel: Maximum homing velocity. Unit: pulse/second.

F64 Tacc: Acceleration time. Unit: second.

I16 AccType: Curve type of acceleration.

0: T-curve

1: S-curve

Return Values:

Example:

```
F64 StrVel = 100;
F64 MaxVel = 100000;
F64 Tacc = 0.1;
I16 AccType = 0;
I16 ret = _8144_home_move(AxisNo, StrVel, MaxVel,
     Tacc, AccType );
//…
I16 HomeStatusInBit;
_8144_home_status(AxisNo, &HomeStatusInBit );
// check home status.
```

See also:

`_8144_home_status`

_8144_home_status

Description:

This function is used to get homing status. When the axis is in homing operation, the “In homing operation” signal (Bit 0) will turned on. When it stopped, “In homing operation” signal (Bit 0) will turned off. If the axis is stopped by error, the “Abnormal stop” (Bit 2) signal will be turned on. Otherwise, “Normal stop” (Bit 1) signal will turned on.

Syntax:

```
I16 _8144_home_status( I16 AxisNo, I16
     *HomeStatusInBit );
B_8144_home_status( ByVal AxisNo As Integer,
     HomeStatusInBit As Integer ) As Integer
```

Parameters:

`I16 AxisNo`: Axis number of Target Axis.

`I16 *HomeStatusInBit`: Status of homing operation.

Bit 0: In homing operation.

Bit 1: Normal stop.

Bit 2: Abnormal stop.

Bit 3-15: Reserved.

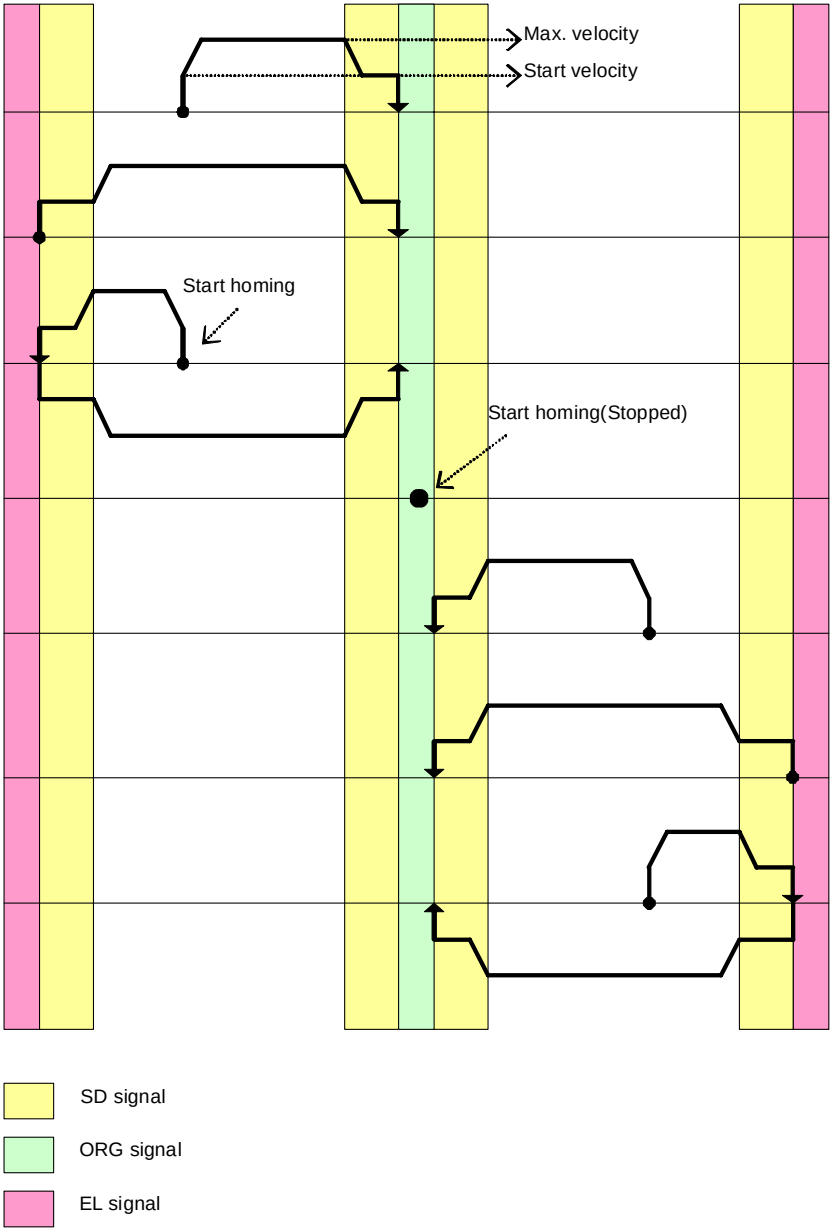
Return Values:

Example:

```
// initialize
// Perform a home return move.
_8144_home_move(AxisNo, StrVel, MaxVel, Tacc,
     AccType );
//…
I16 HomeStatusInBit;
_8144_home_status(AxisNo, &HomeStatusInBit );
if( HomeStatusInBit & 2 ) //Check home move
     status.
{
     //Home move is done.
}
```

See also:

_8144_home_move


Max. velocity
Start velocity
Start homing
Start homing(Stopped)
SD signal
ORG signal
EL signal

6.6 Motion status

_8144_motion_done

Get the motion command to stop or not

Description:

By this function, the user can monitor operation status.

Syntax:

```
I16 _8144_motion_done( I16 AxisNo, I16
     *OperatingOrStop )
B_8144_motion_done( ByVal AxisNo As Integer,
     OperatingOrStop As Integer) As Integer
```

Parameters:

`I16 AxisNo`: The axis number is designated to receive the command.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

`I16 *OperatingOrStop`: The information of wheather the axis is operating or stopped.

- ▷ 0: Stopped(not output pulse)
- ▷ 1: Operating(output pulse)

Return Values:

`ERR_ParametersInvalid`

```
ERR_DeviceNotInitial
ERR_NoError
```

Example:

```
I16 OperatingOrStop = 0;
ret = _8144_start_sr_move( AxisNo, Distance,
      StrVel, MaxVel, Tacc );
while( (!ret) && (!OperatingOrStop) ){
      ret = _8144_motion_done(AxisNo,
      &OperatingOrStop ); // check motion done
}…
```

See also:

```
_8144_motion_status
```

_8144_motion_status

Get motion status

Description:

Return the motion status of the 8144. The information of motion status is in bit format.

The meaning of bit[0] of parameter “MotionStatusInBit” is the same as “_8144_motion_done”

Syntax:

```
I16 _8144_motion_status( I16 AxisNo, I16
     *MotionStatusInBit )
B_8144_motion_status( ByVal AxisNo As Integer,
     MotionStatusInBit As Integer ) As Integer
```

Parameters:

`I16 AxisNo`: The axis number is designated to receive the command.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

`I16 *MotionStatusInBit`:

- bit[0]: operation status. 0: stop, 1: operation
- bit[1]: preset counter. 0: remaining pulse (preset counter) is not zero, 1: is zero
- **bit[2]: 0: remaining pulse > R5, 1: preset counter <= R5**
- bit[3]: 0: Not accelerating, 1: accelerating
- bit[4]: 0: Not decelerating, 1: decelerating
- Else bits: reserved

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_NoError
```

Example:

```
I16 MotionStatusInBit;
I16 ret = _8144_motion_status(AxisNo,
     &MotionStatusInBit );
If( (MotionStatus >> 3) & 0x1)
{…//monitor acceleration status}
```

See also:

```
_8144_motion_done
```

6.7 Interrupt

_8144_set_motion_interrupt_factor

Set factors of motion related interrupts

Description:

This function allows users to enable or disable interrupt output when the motor is stopped in start_r_move or when it is stoped using EL, STP and ORG signals or software stop command. Once the Interrupt function is enabled, you can use **`_8144_wait_single_motion_interruptt()`** to wait event. Disable the motion interrupt by set the factor to 0. Notice that, this function will valid on next motion command.

Syntax:

```
I16 _8144_set_motion_interrupt_factor( I16
      AxisNo, I16 MotionIntFactorInBit )
B_8144_set_motion_interrupt_factor( ByVal AxisNo
      As Integer, ByVal MotionIntFactorInBit As
      Integer ) As Integer
```

Parameters:

`I16 AxisNo`: The axis number is designated to receive the command.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

`I16 MotionIntFactorInBit`: factor of INT

- ▷ 0: disable INT
- ▷ 1: Enable output INT when motion stop

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_InterruptNotEnable
ERR_TimeOut
ERR_Win32Error
ERR_NoError
```

Example:

```
I32 TimeOutMs = 10000; //time out in milisecound
I16 MotionIntFactorBitNum = 0; //bit number = the
     first bit=0
I16 MotionIntFactorInBit = 0x1; //enable motion
     interrupt
ret = _8144_set_motion_interrupt_factor( AxisNo,
     MotionIntFactorInBit );
//…start a move function…
ret = _8144_wait_single_motion_interrupt( AxisNo,
     MotionIntFactorBitNum, TimeOutMs );
if( ret == ERR_NoError ){ //Interrupt occurred!}
I16 MotionIntFactorInBit = 0; //disable
ret = _8144_set_motion_interrupt_factor( AxisNo,
     MotionIntFactorInBit ); //disable motion
     interrupt
```

See also:

```
_8144_wait_single_motion_interrupt
_8144_motion_status
```

_8144_wait_single_motion_interrupt

Wait a single motion related interrupt

Description:

When user enabled the Interrupt function by **_8144_set_motion_int_factor()**. User could use this function to wait the specific interrupt. When this function was running, the process would never stop until evens were triggered or the function was time out. This function returns when one of the following occurs:

1. The specified MotionIntFactorInBit is in the signaled state.
2. The TimeOutMs interval elapses.

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

Syntax:

```
I16 _8144_wait_single_motion_interrupt( I16
     AxisNo, I16 MotionIntFactorBitNum, I32
     TimeOutMs )
B_8144_wait_single_motion_interrupt( ByVal AxisNo
     As Integer, ByVal MotionIntFactorBitNum As
     Integer, ByVal TimeOutMs As Long )As Integer
```

Parameters:

I16 AxisNo: The axis number is designated to receive the command.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3

Card ID	Physical axis	AxisNo
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

I16 MotionIntFactorBitNum: Specifies the bit number of the INT factor.

- ▷ Eg. Stop INT : MotionIntFactorBitNum = 0;

I32 TimeOutMs: Specifies the time-out interval, in milliseconds.

If TimeOut_ms is zero, the function tests the states of the specified objects and returns immediately. If TimeOut_ms is -1, the function's time-out interval never elapses (infinate).

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_InterruptNotEnable
ERR_TimeOut
ERR_Win32Error
ERR_NoError
```

Example:

```
I32 TimeOutMs = 10000; //time out in milisecound
I16 MotionIntFactorBitNum = 0; //bit number = the
     first bit=0
I16 MotionIntFactorInBit = 0x1; //enable motion
     interrupt
ret = _8144_set_motion_interrupt_factor( AxisNo,
     MotionIntFactorInBit );
//…start a move function…
ret = _8144_wait_single_motion_interrupt( AxisNo,
     MotionIntFactorBitNum, TimeOutMs );
if( ret == ERR_NoError ){ //Interrupt occurred!}
I16 MotionIntFactorInBit = 0; //disable
```

```
ret = _8144_set_motion_interrupt_factor( AxisNo,
      MotionIntFactorInBit ); //disable motion
      interrupt
```

See also:

_8144_set_motion_interrupt_factor

_8144_set_gpio_interrupt_factor

Set factors of digital input interrupt

Description:

This function allows users to select GPIO related factors to initiate the event int. Once the Interrupt function is enabled, you can use **_8144_wait_single_gpio_interrupt** or **_8144_wait_multiple_gpio_interrupt** to wait event. To disable the GPIO interrupt function set the GPIO interrupt factor to 0.

Syntax:

```
I16 _8144_set_gpio_interrupt_factor( I16 CardId,
     I32 GpioIntFactorInBit )
B_8144_set_gpio_interrupt_factor( ByVal CardId As
     Integer, ByVal GpioIntFactorInBit As Long
     )As Integer
```

Parameters:

I16 CardId: The card ID number.

I32 GpioIntFactorInBit: The GPIO interrupt factor in bit format

- Bit0 - bit7 : DI0 - DI7 Falling edge
- Bit8 - bit15: DI0 - DI7 Rising edge

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_NoError
```

Example:

```
I32 factorInBits = 0x01; //Wait DI Ch0 falling
     edge
ret = _8144_set_gpio_interrupt_factor( CardId,
     factorInBits );

// wait single GPIO interrupt…
I32 TimeOutMs = 10000; //10 sec timeout
```

```
I16 GpioIntFactorBitNum = 0; //wait INT factor
      bit number 0
ret = _8144_wait_single_gpio_interrupt( CardId,
      GpioIntFactorBitNum, TimeOutMs );
if( ret == ERR_NoError ){
      // Interrupt be triggered
}else{
      //Timeout or wait failed
}
_8144_set_gpio_interrupt_factor( gCardId, 0 ); //
      Disable GPIO interrupt
```

See also:

```
_8144_wait_single_gpio_interrupt,
      _8144_wait_multiple_gpio_interrupt
```

_8144_wait_single_gpio_interrupt

Wait a single digital input interrupt

Description:

When the user enabled the Interrupt function and set the interrupt factors by **_8144_set_gpio_int_factor()**. The user could use this function to wait a specific interrupt. When this function was running, the process would never stop until evens were triggered or the function was time out. This function returns when one of the following occurs:

1. The specified GPIO interrupt factor is in the signaled state.
2. The TimeOutMs interval elapses.

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

Syntax:

```
I16 _8144_wait_single_gpio_interrupt( I16 CardId,
     I16 GpioIntFactorBitNum, I32 TimeOutMs )
B_8144_wait_single_gpio_interrupt(ByVal CardId As
     Integer, ByVal GpioIntFactorBitNum As
     Integer, ByVal TimeOutMs As Long ) As
     Integer
```

Parameters:

I16 CardId: Specify the index of target PCI-8144 card. The card_id could be decided by DIP switch (SW1) or depend on slot sequence.Please refer to "_8144_initial".

I16 GpioIntFactorBitNum: Specifies the bit number of the INT factor

- ▷ Eg. GpioIntFactorBitNum = 10, wait DI_2 rising edge INT factor

I32 TimeOutMs: Specifies the time-out interval, in milliseconds.

If TimeOut_ms is zero, the function tests the states of the specified objects and returns immediately. If TimeOut_ms is -1, the function's time-out interval never elapses (infinate).

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_ParametersInvalid
ERR_InterruptNotEnable
ERR_TimeOut
ERR_Win32Error
ERR_NoError
```

Example:

```
I32 factorInBits = 0x01; //Wait DI Ch0 falling
     edge
ret = _8144_set_gpio_interrupt_factor( CardId,
     factorInBits );

// wait single GPIO interrupt…
I32 TimeOutMs = 10000; //10 sec timeout
I16 GpioIntFactorBitNum = 0; //wait INT factor
     bit number 0
ret = _8144_wait_single_gpio_interrupt( CardId,
     GpioIntFactorBitNum, TimeOutMs );
if( ret == ERR_NoError ){
     // Interrupt be triggered
}else{
     //Timeout or wait failed
}
_8144_set_gpio_interrupt_factor( gCardId, 0 ); //
     Disable GPIO interrupt
```

See also:

```
_8144_wait_multiple_gpio_interrupt
_8144_set_gpio_interrupt_factor
```

_8144_wait_multiple_gpio_interrupt

Wait multiple digital input interrupts

Description:

When the user enabled the Interrupt function and set the interrupt factors by **_8144_set_gpio_int_factor()**. The user could use this function to wait multiple specifid interrupts. When this function was running, the process would never stop until evens were triggered or the function was time out. This function returns when one of the following occurs:

1. Either any one or all of the specified GPIO interrupts are in the signaled state.
2. The TimeOutMs interval elapses.

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

Syntax:

```
I16 _8144_wait_multiple_gpio_interrupt( I16
     CardId, I32 GpioIntFactorInBits, I16
     WaitOption, I32 TimeOutMs,  I32
     *GpioIntTriggeredInBits )
B_8144_wait_multiple_gpio_interrupt( ByVal CardId
     As Integer, ByVal GpioIntFactorInBits As
     Long, ByVal WaitOption As Integer, ByVal
     TimeOutMs As Long, GpioIntTriggeredInBits As
     Long )As Integer
```

Parameters:

I16 CardId: Specify the index of target PCI-8144 card. The card_id could be decided by DIP switch (SW1) or depend on slot sequence.Please refer to “_8144_initial”.

I32 GpioIntFactorInBits:

I16 WaitOption: Specifies the wait option. If WaitOption = 1, the function returns when the state all interrupt factors you specified in GpioIntFactorInBits is signaled. If WaitOption = 0,

the function returns when the state of any one of the interrupt factors you specified in GpioIntFactorInBits is signaled. In the latter case, you can check which factors whose state caused the function to return by compare the parameter “GpioIntTriggeredInBits”.

I32 TimeOutMs: Specifies the time-out interval, in milliseconds.

I32 *GpioIntTriggeredInBits: Information of interrupt state in bit format.

- The value of each bit means: 0: not triggered, 1: be triggered

Return Values:

```
ERR_NoError
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_InsufficientMemory
ERR_InterruptNotEnable
ERR_Win32Error
```

Example:

```
I32 GpioIntFactorInBits = 0x101; //Wait DI Ch0
     falling edge and rising edge
ret = _8144_set_gpio_interrupt_factor( CardId,
     GpioIntFactorInBits);

// wait multiple GPIO interrupts…
I32 TimeOutMs = 10000; //10 sec timeout
I16 WaitOption = 1; //wait all interrupts occured
I32 GpioIntTriggeredInBits;
ret = _8144_wait_multiple_gpio_interrupt(
     gCardId, GpioIntFactorInBits, WaitOption,
     TimeOutMs,  &GpioIntTriggeredInBits );
if( ret == ERR_NoError ){
     // Interrupt be triggered
}else{
     //Timeout or wait failed
}
_8144_set_gpio_interrupt_factor( gCardId, 0 ); //
     Disable GPIO interrupt
```

See also:

```
_8144_wait_single_gpio_interrupt
_8144_set_gpio_interrupt_factor
```

6.8 General purpose input/output

_8144_get_gpio_input

Get all DIN digital input signals

Description:

PCI-8144 has 8 digital input channels. By this function, the user can get the digital input status.

Syntax:

```
I16 _8144_get_gpio_input( I16 CardId, I16
     *DiStatusInBit )
B_8144_get_gpio_input( ByVal CardId As Integer,
     DiStatusInBit As Integer )As Integer
```

Parameters:

`I16 CardId`: Specify the PCI-8144 card index. The CardId could be decided by DIP switch (SW1) or depend on slot sequence.Please refer to _8144_initial()

`I16 *DiStatusInBit`: Digital input status, Bit0-7: Digital Input CH0-7.

- ▷ Meaning of each bit: 1:ON , 0:OFF

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_NoError
```

Example:

```
I16 CardId = 0;
I16 DiStatusInBit;
ret = _8144_get_gpio_input( CardId,
     &DiStatusInBit );
if( ret == ERR_NoError) // compar DiStatusInBit…
```

See also:

```
_8144_get_gpio_input_channel
```

_8144_get_gpio_input_channel

Get a specified DINn digital input signal

Description:

PCI-8144 has 8 digital input channels. By this function, the user can get a specified DINn digital input signal.

Syntax:

```
I16 _8144_get_gpio_input_channel( I16 CardId, I16
     ChannelNum, I16 *DiStatus )
B_8144_get_gpio_input_channel( ByVal CardId As
     Integer, ByVal ChannelNum As Integer,
     DiStatus As Integer ) As Integer
```

Parameters:

`I16 CardId`: Specify the PCI-8144 card index. The CardId could be decided by DIP switch (SW1) or depend on slot sequence.Please refer to _8144_initial()

`I16 ChannelNum`: Specifed the channel number that you want to get the status

- ▷ Value meaning: 0-7 => ch0-ch7

`I16 *DiStatus`: 1: ON, 0: OFF

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_NoError
```

Example:

```
I16 CardId = 0; // Card ID number
I16 ChannelNum = 3; //DI channel 3
I16 DiStatus;
ret = _8144_get_gpio_input_channel (CardId,
     ChannelNum, &DiStatus );
if( ret == ERR_NoError)
{ // if( DiStatus == 1) …}
```

See also:

```
_8144_get_gpio_input
```

_8144_set_gpio_output

Set all DOUT digital output signal

Description:

The PCI-8144 has 8 digital output channels. By this function, the user could control all digital outputs.

Syntax:

```
I16 _8144_set_gpio_output( I16 CardId, I16
      DoValueInBit )
B_8144_set_gpio_output( ByVal CardId As Integer,
      ByVal DoValueInBit As Integer)As Integer
```

Parameters:

`I16 CardId`: Specify the PCI-8144 card index. The CardId could be decided by DIP switch (SW1) or depend on slot sequence.Please refer to _8144_initial()

`I16 DoValueInBit`: bit0 – bit7 => DO CH0-7

- ▷ The meaning of each bit: 0:OFF, 1:ON

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_NoError
```

Example:

```
I16 CardId = 0; //Card ID number
I16 DoValueInBit = 0xFF; //turn ON all DO channal
ret = _8144_set_gpio_output(CardId, DoValueInBit
      );
…
```

See also:

```
_8144_set_gpio_output_channel
_8144_get_gpio_output
_8144_get_gpio_output_channel
```

_8144_set_gpio_output_channel

Set a specified DOUTn digital output signal

Description:

The PCI-8144 has 8 digital output channels. By this function, the user could control a specified digital output.

Syntax:

```
I16 _8144_set_gpio_output_channel( I16 CardId,
     I16 ChannelNum, I16 DoValue )
B_8144_set_gpio_output_channel( ByVal CardId As
     Integer, ByVal ChannelNum As Integer, ByVal
     DoValue As Integer )As Integer
```

Parameters:

`I16 CardId`: Specify the PCI-8144 card index. The CardId could be decided by DIP switch (SW1) or depend on slot sequence.Please refer to _8144_initial()

`I16 ChannelNum`: Specifed the channel number that you want to control

- ▷ Value meaning: 0-7 => ch0-ch7

`I16 DoValue`: 0:OFF, 1: ON

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_NoError
```

Example:

```
I16 CardId = 0; // Card ID number
I16 ChannelNum = 0; //DO channel 0
I16 DoValue = 1; //ON
ret = _8144_set_gpio_output_channel( I16 CardId,
     I16 ChannelNum, I16 DoValue ); …
```

See also:

```
_8144_set_gpio_output
_8144_get_gpio_output
_8144_get_gpio_output_channel
```

_8144_get_gpio_output

Get all DOUT digital output signal

Description:

Get all DOUT digital output signal.

Syntax:

```
I16 _8144_get_gpio_output( I16 CardId, I16
     *DoValueInBit )
B_8144_get_gpio_output( ByVal CardId As Integer,
     DoValueInBit As Integer)As Integer
```

Parameters:

`I16 CardId`: Specify the PCI-8144 card index. The CardId could be decided by DIP switch (SW1) or depend on slot sequence.Please refer to _8144_initial().

`I16 *DoValueInBit`: The information of DO signals, bit0 – bit7 => DO CH0-7

- ▷ The meaning of each bit: 0:OFF, 1:ON

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_NoError
```

Example:

```
I16 CardId = 0; //Card ID number
I16 DoValueInBit;
ret = _8144_get_gpio_output(CardId, &DoValueInBit
     );
if( ret == ERR_NoError )
{// compare DoValueInBIt…}
```

See also:

```
_8144_set_gpio_output
_8144_set_gpio_output_channel
_8144_get_gpio_output_channel
```

_8144_get_gpio_output_channel

Get a specified DOUTn digital output signal

Description:

Syntax:

```
I16 _8144_get_gpio_output_channel( I16 CardId,
      I16 ChannelNum, I16 *DoValue )
B_8144_get_gpio_output_channel( ByVal CardId As
      Integer, ByVal ChannelNum As Integer,
      DoValue As Integer ) As Integer
```

Parameters:

`I16 CardId`: Specify the PCI-8144 card index. The CardId could be decided by DIP switch (SW1) or depend on slot sequence.Please refer to _8144_initial().

`I16 ChannelNum`: Specified the channel number of digital output.

- ▷ CH0-CH7 : 0-7

`I16 *DoValue`: The information of the specified DO signal.

- ▷ Value meaning: 0:OFF, 1:ON

Return Values:

```
ERR_ParametersInvalid
ERR_DeviceNotInitial
ERR_NoError
```

Example:

```
I16 CardId = 0; //Card ID number
I16 ChannelNum = 0; //DO ch0
I16 DoValue; //Do information
ret = _8144_get_gpio_output_channel( CardId,
      ChannelNum, &DoValue );
…
```

See also:

```
_8144_set_gpio_output
_8144_set_gpio_output_channel
_8144_get_gpio_output
```

6.9 Speed profile calculation

_8144_get_tv_move_profile

Get tv_move speed profile

Description:

This function is used to get the tv_move speed profiles. By this function, user can get the actual speed profile before running.

This function has warnning return code. please refer to **`_8144_tv_move function`**.

Syntax:

```
I16 _8144_get_tv_move_profile( I16 AxisNo, F64
      StrVel, F64 MaxVel, F64 Tacc, F64 *StrVelP,
      F64 *MaxVelP, F64 *TaccP, F64 *MagnP )
B_8144_get_tv_move_profile( ByVal AxisNo As
      Integer, ByVal StrVel As Double, ByVal
      MaxVel As Double, ByVal Tacc As Double,
      StrVelP As Double, MaxVelP As Double, TaccP
      As Double, MagnP As Double ) As Integer
```

Parameters:

`I16 AxisNo`: Axis number of Target Axis.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

`F64 StrVel`: Start velocity (pulse/sec)

F64 MaxVel: Maximum velocity (pulese/sec)

F64 Tacc: Acceleration time (sec)

F64 *StrVelP: The information of calculated start velocity (pulse/sec)

F64 *MaxVelP: The information of maximum velocity (pulse/sec)

F64 *TaccP: The information of acceleration time (sec)

F64 *MagnP: Set to 0

Return Values:

```
ERR_ParametersInvalid
WAR_AccDecTimeToSmall
WAR_AccDecTimeToLarge
ERR_NoError
```

Example:

```
I16 ret; //return code
I16 AxisNo = 0; //axis number
F64 StrVel = 100.0; //start velocity (pps)
F64 MaxVel = 50000.0; //maximum velocity (pps)
F64 Tacc = 0.5; // acceleration time (sec)
F64 *StrVelP, MaxVelP, TaccP; //The speed profile
     informations
ret = _8144_get_tv_move_profile(AxisNo, StrVel,
     MaxVel, Tacc, &StrVelP, &MaxVelP, &TaccP, 0
     );
```

See also:

```
_8144_tv_move()
_8144_get_sv_move_profile()
```

_8144_get_sv_move_profile

Get sv_move speed profile

Description:

This function is used to get the sv_move speed profiles. By this function, user can get the actual speed profile before running.

This function has warnning return code. please refer to **_8144_tv_move function**.

Syntax:

```
I16 _8144_get_sv_move_profile( I16 AxisNo, F64
     StrVel, F64 MaxVel, F64 Tacc, F64 *StrVelP,
     F64 *MaxVelP, F64 *TaccP, F64 *MagnP )
B_8144_get_sv_move_profile( ByVal AxisNo As
     Integer, ByVal StrVel As Double, ByVal
     MaxVel As Double, ByVal Tacc As Double,
     StrVelP As Double, MaxVelP As Double, TaccP
     As Double, MagnP As Double ) As Integer
```

Parameters:

I16 AxisNo: Axis number of Target Axis.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

F64 StrVel: Start velocity (pulse/sec)

F64 MaxVel: Maximum velocity (pulese/sec)

F64 Tacc: Acceleration time (sec)

F64 *StrVelP: The information of calculated start velocity (pulse/sec)

F64 *MaxVelP: The information of maximum velocity (pulse/sec)

F64 *TaccP: The information of acceleration time (sec)

F64 *MagnP: Set to 0

Return Values:

```
EERR_ParametersInvalid
WAR_AccDecTimeToSmall
WAR_AccDecTimeToLarge
ERR_NoError
```

Example:

```
I16 ret; //return code
I16 AxisNo = 0; //axis number
F64 StrVel = 100.0; //start velocity (pps)
F64 MaxVel = 50000.0; //maximum velocity (pps)
F64 Tacc = 0.5; // acceleration time (sec)
F64 *StrVelP, MaxVelP, TaccP; //The speed profile
     informations
ret = _8144_get_sv_move_profile(AxisNo, StrVel,
     MaxVel, Tacc, &StrVelP, &MaxVelP, &TaccP, 0
     );
```

See also:

```
_8144_sv_move()
_8144_get_tv_move_profile()
```

_8144_get_start_tr_move_profile

Get start_tr_move speed profile

Description:

This function is used to get the relative trapezoidal speed profiles. By this function, user can get the actual speed profile before running.

This function has warnning return code. please refer to **_8144_tv_move function**.

Syntax:

```
I16 _8144_get_start_tr_move_profile( I16 AxisNo,
      F64 Distance, F64 StrVel, F64 MaxVel, F64
      Tacc, F64 *StrVelP, F64 *MaxVelP, F64
      *TaccP, F64 *TConstP, F64 *MagnP )
B_8144_get_start_tr_move_profile( ByVal AxisNo As
      Integer, ByVal Distance As Double, ByVal
      StrVel As Double, ByVal MaxVel As Double,
      ByVal Tacc As Double, StrVelP As Double,
      MaxVelP As Double, TaccP As Double, TconstP
      As Double, MagnP As Double ) As Integer
```

Parameters:

I16 AxisNo: Axis number of Target Axis.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

F64 Distance: Specified relative distance to move (pulse)

F64 StrVel: Starting velocity (pulse/sec)

F64 MaxVel: Maximum velocity (pulse/sec)

F64 Tacc: Specified acceleration and deceleration time (sec)

F64 *StrVelP: Actual start velocity (pulse/sec)

F64 *MaxVelP: Actual Maximum velocity (pulse/sec)

F64 *TaccP: Actual acceleration and deceleration time (sec)

F64 *TconstP: constant speed time(maximum velocity)

F64 *MagnP: Set to 0

Return Values:

```
EERR_ParametersInvalid
WAR_AccDecTimeToSmall
WAR_AccDecTimeToLarge
ERR_NoError
```

Example:

```
I16 ret; //return code
I16 AxisNo = 0; //axis number
F64 Distance =100000.0 ; //pulse number
F64 StrVel = 100.0; //start velocity (pps)
F64 MaxVel = 50000.0; //maximum velocity (pps)
F64 Tacc = 0.5; // acceleration time (sec)
F64 *StrVelP, MaxVelP, TaccP, TConstP; //The
     speed profile informations
ret = _8144_get_start_tr_move_profile(AxisNo,
     Distance, StrVel, MaxVel, Tacc, &StrVelP,
     &MaxVelP, &TaccP, &TConstP, 0 );
…
```

See also:

```
_8144_start_tr_move
_8144_get_start_sr_move_profile
```

_8144_get_start_sr_move_profile

Get start_sr_move speed profile

Description:

This function is used to get the relative S-curve speed profiles. By this function, user can get the actual speed profile before running.

This function has warnning return code. please refer to **_8144_sv_move function**.

Syntax:

```
I16 _8144_get_start_sr_move_profile( I16 AxisNo,
      F64 Distance, F64 StrVel, F64 MaxVel, F64
      Tacc, F64 *StrVelP, F64 *MaxVelP, F64
      *TaccP, F64 *TConstP, F64 *MagnP )
B_8144_get_start_sr_move_profile( ByVal AxisNo As
      Integer, ByVal Distance As Double, ByVal
      StrVel As Double, ByVal MaxVel As Double,
      ByVal Tacc As Double, StrVelP As Double,
      MaxVelP As Double, TaccP As Double, TconstP
      As Double, MagnP As Double ) As Integer
```

Parameters:

I16 AxisNo: Axis number of Target Axis.

Card ID	Physical axis	AxisNo
0	0	0
	1	1
	2	2
	3	3
1	0	4
	1	5
	2	6
	3	7
2	0	8
	...	...

F64 Distance: Specified relative distance to move (pulse)

F64 StrVel: Starting velocity (pulse/sec)

F64 MaxVel: Maximum velocity (pulse/sec)

F64 Tacc: Specified acceleration and deceleration time (sec)

F64 *StrVelP: Actual start velocity (pulse/sec)

F64 *MaxVelP: Actual Maximum velocity (pulse/sec)

F64 *TaccP: Actual acceleration and deceleration time (sec)

F64 *TconstP: constant speed time(maximum velocity)

F64 *MagnP: Set to 0

Return Values:

```
EERR_ParametersInvalid
WAR_AccDecTimeToSmall
WAR_AccDecTimeToLarge
ERR_NoError
```

Example:

```
I16 ret; //return code
I16 AxisNo = 0; //axis number
F64 Distance =100000.0 ; //pulse number
F64 StrVel = 100.0; //start velocity (pps)
F64 MaxVel = 50000.0; //maximum velocity (pps)
F64 Tacc = 0.5; // acceleration time (sec)
F64 *StrVelP, MaxVelP, TaccP, TConstP; //The
     speed profile informations
ret = _8144_get_start_sr_move_profile(AxisNo,
     Distance, StrVel, MaxVel, Tacc, &StrVelP,
     &MaxVelP, &TaccP, &TConstP, 0 );
…
```

See also:

```
_8144_start_sr_move
_8144_get_start_tr_move_profile
```

7 Function Return Code

The following table provides a list of possible return value in our library. If the return value is not zero, it means there are some error or warning occurred.

Error Code

Code	Define	Describtion
0	ERR_NoError	No Error, function success
-1	ERR_OSVersion	Operation System type mismatched
-2	ERR_OpenDriverFailed	Open device driver failed - Create driver interface failed
-3	ERR_InsufficientMemory	System memory insufficiently
-4	ERR_DeviceNotInitial	Cards not be initialized
-5	ERR_NoDeviceFound	Cards not found(No card in your system)
-6	ERR_CardIdDuplicate	Cards' ID Number duplicate
-7	ERR_DeviceAlreadyInitialed	Cards have been initialed
-8	ERR_InterruptNotEnable	Cards' interrupt events not enable
-9	ERR_TimeOut	Function time-out
-10	ERR_ParametersInvalid	Function input parameters are invalid
-11	ERR_SetEEPROM	Set data to EEPROM failed
-12	ERR_GetEEPROM	Get data from EEPROM failed
-1000~	ERR_Win32Error	Check Win32 ErrorCode define [win32 error code = -(code + ERR_Win32Error)]

Warning Code

Code	Define	Describtion
1	WAR_AccDecTimeTooLarge	Tacc/Tdec's value is too large
2	WAR_AccDecTimeTooSmall	Tacc/Tdec's value is too small
3	WAR_RampDownPointExceed	The ramping down value is biger than it’s maximum value