

PCI-8134

4 Axes Servo / Stepper

Motion Control Card

Sample Application / ISaGRAF Library

Programming Guide

@Copyright 1999 ADLink Technology Inc.

All Rights Reserved.

Manual Rev. 1.00: September 21, 2000

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.

This document contains proprietary information protected by copyright. All rights are reserved. No part of this manual may be reproduced by any mechanical, electronic, or other means in any form without prior written permission of the manufacturer.

Trademarks

NuDAQ, PCI-8134 are registered trademarks of ADLink Technology Inc, MS-DOS & Windows 95, Windows 98, Windows NT, Visual Basic, Visual C++ are registered trademarks of Microsoft Corporation, Borland C++ is a registered trademark of Borland International, Inc. Other 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 always the most important thing for ADLINK Tech Inc. If you need any help or service, please contact us and get it.

ADLINK Technology Inc.			
Web Site	http://www.adlink.com.tw http://www.adlinktechnology.com		
Sales & Service	service@Adlink.com.tw		
Technical Support	NuDAQ	nudaq@adlink.com.tw	
	NuDAM	nudam@adlink.com.tw	
	NuIPC	nuipc@adlink.com.tw	
	NuPRO	nupro@adlink.com.tw	
	Software	sw@adlink.com.tw	
TEL	+886-2-82265877	FAX	+886-2-82265717
Address	9F, No. 166, Jian Yi Road, Chungho City, Taipei, 235 Taiwan,		

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

Detailed Company Information			
Company/Organization			
Contact Person			
E-mail Address			
Address			
Country			
TEL		FAX	
Web Site			

Questions	
Product Model	
Environment to Use	<input type="checkbox"/> OS: _____ <input type="checkbox"/> Computer Brand: _____ <input type="checkbox"/> M/B: <input type="checkbox"/> CPU: _____ <input type="checkbox"/> Chipset: <input type="checkbox"/> Bios: <input type="checkbox"/> Video Card: <input type="checkbox"/> Network Interface Card: <input type="checkbox"/> Other:
Challenge Description	
Suggestions for ADLINK	

Contents

Chapter 1 Introduction	1
Chapter 2 Quick-Reference Guide	2
2.1 PCI-8134 Configuration.....	2
2.2 Use Configuration Utility.....	2
2.3 Simple Function Test	2
2.4 Motion Done Status & I/O Status Monitoring	2
2.5 Interrupt Handling under Windows 95/98 by Visual Basic 5.0.....	3
2.6 Interrupt Handling under Windows 95/98 by Visual C++ 6.0.....	3
2.7 Interrupt Handling under DOS with C	3
2.8 Position Control by various types of Velocity Profiles.....	3
2.9 Homing routines.....	3
2.10 Multiple Axes Synchronized Motion	4
2.11 Linear and Circular Interpolation.....	4
2.12 Jog.....	4
2.13 Velocity Change On The Fly.....	4
Chapter 3 Examples.....	5
3.1 PCI-8134 Initialization	5
3.1.1 Use 8134.INI to Configure	5
3.1.2 An Example to Load INI file and Configure PCI-8134 ..	7
3.1.3 Programming under Windows NT	9
3.2 Use Configuration Utility.....	9
3.2.1 Config.exe Utility	10
3.2.2 Create an Configuration Interface	11
3.3 Simple Function Test.....	12
3.3.1 Simple Function Test	12
3.4 Motion Done Status and I/O Monitoring	16
3.4.1 Program organization	16
3.4.2 Reading Motion Done Status.....	18
3.4.3 Reading I/O Status.....	18
3.4.4 Relative Motion Button	19
3.5 Interrupt Handling under Windows 95/98 by Visual Basic 5.0.....	20

3.5.1	Interrupt Event	20
3.5.2	Create an Event	20
2.5.3	Create a Thread.....	20
2.5.4	Use WIN32 API in Visual Basic	21
2.5.5	A Complete Example	22
3.6	Interrupt Handling under Windows 95/98 by Visual C++	
6.0.....	25
3.6.1	Use PCI-8134 function library in Visual C++	25
3.6.2	Create a Dialog-based MFC project	25
3.7	Interrupt Handling under DOS	28
3.7.1	A Skeleton Program for Interrupt Handling.....	28
3.7.2	Other Empty ISR functions	30
3.8	Position Control by various types of Velocity Profiles... 31	
3.8.1	Velocity Profile.....	31
3.8.2	Various types of Velocity Profile for Position Control	
.....	31
3.9	Homing routines.....	35
3.9.1	Basic Homing Styles.....	35
3.9.2	Advanced Homing Styles.....	37
3.9.3	Implement Homing Program.....	39
3.10	Multiple Axes Synchronized Motion	44
3.10.1	How to use start_move_all()	44
3.10.2	Implement Multiple Axes Synchronized Motion.....	44
3.11	Linear and Circular Interpolation.....	47
3.11.1	How to use Linear Interpolation Functions.....	47
3.11.2	How to use Circular Interpolation Functions	48
3.11.3	Coordinate System in Microsoft Windows®	48
3.11.4	DC (Device Context).....	51
3.11.5	Animation	52
3.11.6	Implement 2-D example	55
3.12	Jog.....	58
3.12.1	Create a Thread for Jogging	59
3.12.2	Create a Scope for Display Jogging	61
3.13	Velocity Change On The Fly.....	61
3.13.1	Velocity Change on the fly.....	61
3.13.2	Velocity Value	62
3.13.3	Velocity Change on the Fly Demo results.....	62
2.13.4	Limitation of Velocity Change on the Fly.....	65
3.13.5	Repeat Mode.....	66
Chapter 4	The ISaGRAF Library for PCI-8134	68

4.1	Installation of PCI-8134 ISaGRAF Library	68
4.1.1	PCI-8134 ISaGRAF Library Installation	68
4.1.2	PCI-8134 ISaGRAF Library Un-installation.....	69
4.2	Restore PCI-8134 ISaGRAF Library C Function Objects in the ISaGRAF Workbench.....	69
3.2.1	With ADLink's "PCI-8134 ISaGRAF C function Object" diskettes.....	69
3.2.2	With "ADLink All-In-One Compact Disc":	69
4.3	Restore PCI-8134 ISaGRAF Sample Programs.....	70
4.3.1	With ADLink's "PCI-8134 ISaGRAF Sample Program" diskettes	70
4.3.2	With "ADLink All-In-One Compact Disc":	70
4.4	The definition of PCIS-ISG 8134 ISaGRAF Library	71
3.4.1	Initialization function group	71
4.4.2	Pulse Input /Output Configuration function group....	73
4.4.3	Continuously Motion Move function group.....	74
4.4.4	Trapezoidal Motion Mode function group	75
4.4.5	S-Curve Profile Motion function group	77
4.4.6	Linear and Circular Interpolated Motion function group	79
4.4.7	Interpolation Parameters Configuring function group 80	
4.4.8	Interpolation Parameters Configuring function group 81	
4.4.9	Manual Pulser Motion function group.....	82
4.4.10	Motion Status function group	83
4.4.11	Servo Drive Interface function group.....	84
4.4.12	I/O Control and Monitoring function group	85
4.4.13	Position Control function group	86
4.4.14	Interrupt Control function group.....	87
4.5	The mapping between PCIS-8134 NT DLL function and PCI-8134 ISaGRAF Library.....	89

Introduction

In addition to the extensive library of standard functions, ADLink provides an Sample Disk containing a variety of standard applications that you can use as a starting point for applications development. This manual is a companion to that disk.

This document is divided into two main sections. The first is a quick reference that groups the programs into main categories and then lists and describes the individual programs on the disk that fall into that category. The second section is a printout of the sample code files. A cross-reference table of all the files is located at the beginning of this section.

The various application function described here can span a wide variety of customer applications. Look for examples that may be useful to your particular motion control application.

The individual files show internals of the features work. Use these examples as "starter" code to integrate into your development. They will also help you gain an understanding of how the motion card works and how it implements the motion functions.

You can also use these functions to debug individual features or functions and to ensure the motion control subsystem is functioning properly. This can help isolate problems to specific subsystem level.

Quick-Reference Guide

2.1 PCI-8134 Configuration

Demonstrate how to configure axes by 8134.ini and how to create an application project in Visual Basic programming environment.

2.2 Use Configuration Utility

Demonstrate how to call the configuration utility by shell command in user's program.

2.3 Simple Function Test

Demonstrate how to make a simple test by using continuous movement function and by reading position counter feedback.

2.4 Motion Done Status & I/O Status Monitoring

Demonstrate how to read the motion done status and I/O status of an axis. Use the color of text box control item as an indicator to display each status.

2.5 Interrupt Handling under Windows 95/98 by Visual Basic 5.0

Demonstrate how to write a thread to capture the interrupt events triggered by hardware interrupt in Visual Basic 5.0.

2.6 Interrupt Handling under Windows 95/98 by Visual C++ 6.0

Demonstrate how to write a thread to capture the interrupt event triggered by hardware interrupt in Visual C++ MFC.

2.7 Interrupt Handling under DOS with C

Demonstrate how to write an ISR to capture the interrupt triggered by hardware in DOS environment.

2.8 Position Control by various types of Velocity Profiles

Demonstrate how to write a program for a particular type of position control by choosing a correct function.

2.9 Homing routines

Demonstrate 6 types of homing routines and explain each type's hardware setup. These are as follows:

- (1). Home Mode 0 (provide by PCI-8134 function library)
- (2). Home Mode 1 (provide by PCI-8134 function library)
- (3). Home Mode 2 (provide by PCI-8134 function library)
- (4). Two-stage homing
- (5). Midpoint Homing between positive and negative limits
- (6). Auto Home search

2.10 Multiple Axes Synchronized Motion

Demonstrates how to write a program for multiple axes synchronized motion.

2.11 Linear and Circular Interpolation

Demonstrate how to use the interpolation function sets and how to visualize the 2-D motion by creating a dynamic scope. The DC drawing concept, coordinate transformation, and animation method are introduced here.

2.12 Jog

Demonstrate how to use software method to make a jogging function. The jogging path is displayed in a dynamic scope.

2.13 Velocity Change On The Fly

Demonstrate how to use velocity change function and how to display velocity and position data graphically. The position profile and velocity profile are displayed by a software dynamic scope. User can use other motion types and see the differences from each type of position control profiles.

The repeat mode by software is introduced here, too.

3

Examples

3.1 PCI-8134 Initialization

3.1.1 Use 8134.INI to Configure

We provide a very convenient way to modify your configuration from a Windows® standard INI file. You can edit this file directly by any text editor or use Config.exe from Appendix C to configure this file.

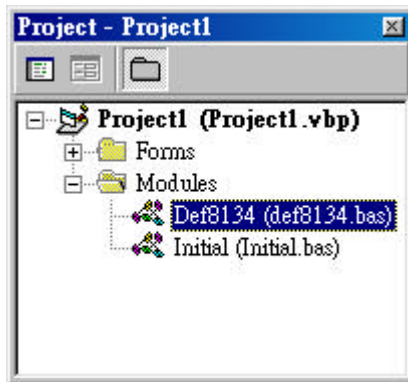
There are several functions and modules need to be added in your VB project before programming.

Please check:

- (1). PCI-8134 card has been inserted in one PCI slot properly
- (2). Make sure that 8134.DLL does exist in your Windows system directory.
- (3). Add two files in your project: Initial.BAS & Def8134.BAS from Appendix A and B

Note: Initial.BAS provides several functions to deal with 8134.INI and Def8134.BAS contents all 8134-function library declarations.

After Add these two files, your project window in VB IDE is like as follows



These two files are very important for 8134 application programming. The source codes of these two files are listed in appendix A & B

An example of 8134.INI files is as follows:

```
[Axis 0]
PLS_OUTMODE= 0
PLS_IPTMODE= 0
CNT_SRC= 0
RATIO= 1
HOME_MODE= 0
ORG_LOGIC= 1
ORG_LATCH= 0
EZ_LOGIC= 0
IPT_MODE= 0
OP_MODE= 0
ALM_LOGIC= 0
ALM_MODE= 0
INP_LOGIC= 0
INP_ENABLE= 0
SD_LOGIC= 1
SD_LATCH= 1
SD_ENABLE= 1
ERC_ENABLE= 0
INT_FACTOR= 0

[Axis 1]
PLS_OUTMODE= 0
PLS_IPTMODE= 2
```

```
CNT_SRC= 1
RATIO= 2.5
HOME_MODE= 0
ORG_LOGIC= 1
ORG_LATCH= 0
EZ_LOGIC= 1
IPT_MODE= 0
OP_MODE= 0
ALM_LOGIC= 0
ALM_MODE= 0
INP_LOGIC= 0
INP_ENABLE= 0
SD_LOGIC= 1
SD_LATCH= 1
SD_ENABLE= 0
ERC_ENABLE= 0
INT_FACTOR= 8642799
```

[Axis2]

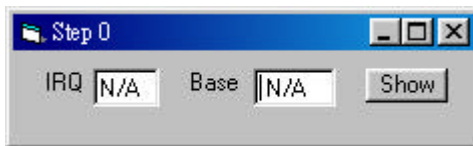
.
.
.

3.1.2 An Example to Load INI file and Configure PCI-8134

Example1: Step by Step

- (1). Open a new project.
- (2). Add two module files: Def8134.BAS and Initial.BAS
- (3). Write Form_Load Procedure
- (4). Add the necessary Form variables
- (5). Add two text boxes and one command button to show IRQ and base address
- (6). Run it

When we press the show button, it will display IRQ number and Base address on text box. This form is like as follows:



In Step 3) Form_Load Procedure

```
Private Sub Form_Load()  
  
    ' Initialize 8134  
    If W_8134_Initial(TotalCard, MyPCI) Then  
        MsgBox "You Don't Have any PCI-8134 Card!"  
    End  
    End If  
  
    'Load INT file and Configure All Setting  
    If LoadConfigFile(Axis, TotalCard) Then  
        MsgBox "You Must Edit an Config File First!"  
    Else  
        ConfigAll Axis, TotalCard  
    End If  
End Sub
```

In Step 4) Add general declaration variables in the Form

```
Dim TotalCard As Integer  
Dim MyPCI As PCI_INFO  
Dim Axis(0 To 4*MAX_PCI_CARDS - 1) As AxisConfig
```

In Step 5) Add some additional objects

Add two text boxes and one button to show IRQ and Base Address Information.

Object Type	Attribute	Value
Command Button	Name	Command1
	Caption	Show
Text Box	Name	T_IRQ
	Text	N/A
Text Box	Name	T_Base
	Text	N/A

The codes in Command1 button:

```
Private Sub Command1_Click()  
Dim CardNo, irqNo, baseAddr as integer  
  
    CardNo=0  
    'Show IRQ and Base Address  
    W_8134_Get_IRQ_Channel CardNo, irqNo  
    W_8134_Get_Base_Addr CardNo, baseAddr  
    T_IRQ.Text = Str(irqNo)  
    T_Base.Text = Hex(baseAddr)  
  
End Sub
```

After all steps above, run it and you will see the results. If it tells that you must edit a configuration file first, you should run the config.exe in the sample disk to create a 8134.INI file.

3.1.3 Programming under Windows NT

There is a little difference when you want to use PCI-8134 library under NT. The Initializing function parameter is different as follows:

```
Int i=0;
U16 Total_Card=0;
For(i=0;i<MAX_PCI_CARDS;i++) {
    If( W_8134_Initial(i) == 0 )
        Total_Card++;
}
```

You can get the card amounts in Total_Card variable.

Finally, you must close PCI-8134 resources when you exit the program. The codes are as follows:

```
int i=0;
for(i=0;i<Total_Card;i++) W_8134_Close(i);
```

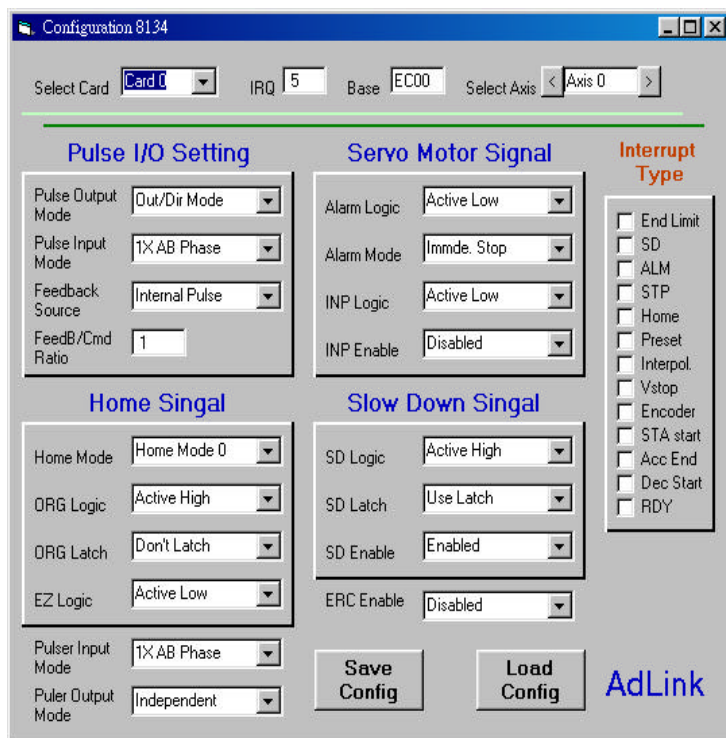
That's all the differences between Win95 and NT programming

3.2 Use Configuration Utility

Sometimes, user needs a configuration interface in his program. We provide a utility – Config.exe to help user to make it. The config.exe can be loaded by “Shell” command in user’s program. Use this program to create or modify 8134.INI and reconfigure PCI-8134 according to this file in user’s program.

3.2.1 Config.exe Utility

The config.exe file is at Appendix C and it looks like as follows:

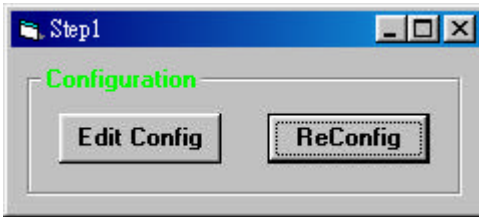


3.2.2 Create an Configuration Interface

Example 2-2: Step by Step

- (1). Open a new project.
- (2). Add two module files: Def8134.BAS and Initial.BAS
- (3). Write Form_Load Procedure
- (4). Add the necessary Form variables
- (5). Add two command buttons and one frame
- (6). Write command button procedure
- (7). Run it

When user press "Edit Config" button, the config.exe will be started. After configure all parameters of each axis, press "Save Config" button in config.exe to save 8134.INI. Finally, press "ReConfig" button in this program to update new parameters from 8134.INI file.



In Step 3) & 4) Please refer to 2.1.2

In Step 5) Add some additional objects

Object Type	Attribute	Value
Command Button	Name	C_ShINI
	Caption	Edit Config
Command Button	Name	C_RCfg
	Caption	ReConfig

In Step 6) The codes for these two buttons are as follows:

```
Private Sub C_RCfg_Click()  
    LoadConfigFile Axis, TotalCard  
    ConfigAll Axis, TotalCard  
End Sub  
  
Private Sub C_ShINI_Click()  
    Shell App.Path & "\" & "Config.exe", vbNormalFocus  
End Sub
```

Notice that there is a Shell function. It loads an executable file in your application's path. So you must copy config.exe file to a correct location or your program won't find this file to load.

The source code of config.exe is in Appendix C

3.3 Simple Function Test

In this section, we will show how to combine the stuffs we have learned in previous sections and add some new features in the program. For example: display position, set continuous movement parameters, and stop button. You can simulate this movement by setting feedback source as command input(Internal Pulse type) instead of connecting any real encoder.

3.3.1 Simple Function Test

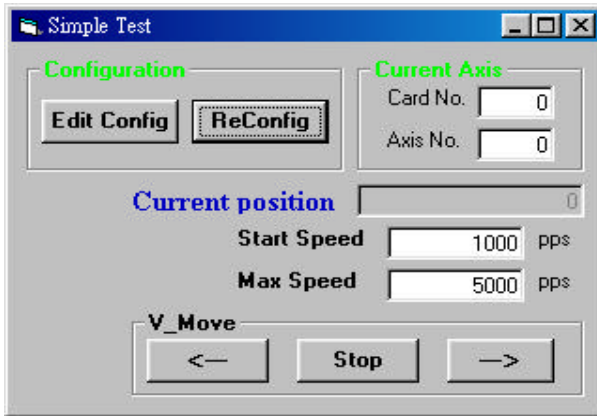
When you press forward or backward button, you will see the number increased or decreased in current position text box. You can press stop button at any moment when you want to stop motion.

Example: Step by Step

- (1). Open a new project.
- (2). Add two module files: Def8134.BAS and Initial.BAS
- (3). Write Form_Load Procedure
- (4). Add the necessary Form variables
- (5). Add buttons and text Boxes
- (6). Write forward/backward procedure
- (7). Write stop button procedure
- (8). Create a timer to get position
- (9). Run it

When user press forward or backward button, the desired axis will move continuously. When user press stop button, the axis will stop immediately. There is a text box to display current pulse amount (current position). There are two text boxes for user to enter the desired card number and axis number. There are also two text boxes for user to enter moving parameters. This program looks like as follows:

In Step 1) to 4) please refer to previous sections



In Step 5) Add buttons and text boxes

Object Type	Attribute	Value
Form*	Name	Form1
	Caption	Simple Test
Command Button*	Name	C_ShINI
	Caption	Edit Config
Command Button*	Name	C_RCfg
	Caption	ReConfig
Command Button	Name	B_Forward
	Caption	<-
Command Button	Name	B_Backward
	Caption	->
Command Button	Name	B_Stop
	Caption	Stop
Text Box	Name	T_CardNo
	Text	0
Text Box	Name	T_AxisNo
	Text	0
Text Box	Name	T_VStr
	Text	1000
Text Box	Name	T_VMax
	Text	5000
Text Box	Name	T_CurPos
	Text	0
Timer	Name	Timer1
	Interval	10

In Step 6) Write Move Forward/Backward procedure

```
Private Sub B_Backward_Click()  
Dim AxisNo As Integer  
Dim CardNo As Integer  
Dim Channel As Integer  
  
    AxisNo = CInt(T_AxisNo.Text)  
    CardNo = CInt(T_CardNo.Text)  
    Channel = 4 * CardNo + AxisNo  
  
    v_move Channel, -Cdbl(T_VStr.Text), -Dbl(T_VMax.Text),  
        0.1  
End Sub  
  
Private Sub B_Forward_Click()  
Dim AxisNo As Integer  
Dim CardNo As Integer  
Dim Channel As Integer  
  
    AxisNo = CInt(T_AxisNo.Text)  
    CardNo = CInt(T_CardNo.Text)  
    Channel = 4 * CardNo + AxisNo  
  
    v_move Channel, Cdbl(T_VStr.Text), Cdbl(T_VMax.Text),  
        0.1  
End Sub
```

In Step 7) Write Stop button procedure

```
Private Sub B_Stop_Click()  
Dim AxisNo As Integer  
Dim CardNo As Integer  
Dim Channel As Integer  
  
    AxisNo = CInt(T_AxisNo.Text)  
    CardNo = CInt(T_CardNo.Text)  
    Channel = 4 * CardNo + AxisNo  
  
    v_stop Channel, 0.1  
End Sub
```

In Step 8) Create a Timer for Displaying position

```
Private Sub Timer1_Timer()  
Dim AxisNo As Integer  
Dim CardNo As Integer  
Dim Channel As Integer  
Dim Pos As Double  
  
    AxisNo = CInt(T_AxisNo.Text)  
    CardNo = CInt(T_CardNo.Text)  
    Channel = 4 * CardNo + AxisNo  
  
    get_position Channel, Pos  
    T_CurPos.Text = Str(Pos)  
End Sub
```

Note: If these programs don't work, try to check your parameter settings by using Config.exe or to check your hardware.

3.4 Motion Done Status and I/O Monitoring

Motion status or motion done status is an integer value from 0 to 5 which is returned from PCI-8134. It tells user the motion status of one axis. Please refer to the table in section 2.4.2

There are 12 I/O statuses in each axis. PCI-8134 will return an integer value to represent these statuses. Please refer to the table in section 2.4.3

3.4.1 Program organization

We add a relative movement function and two rows of text boxes to indicate I/O and motion status. When the specific I/O status is active, the color of the respective text box turns to green. User can test hardware I/O point by this program.

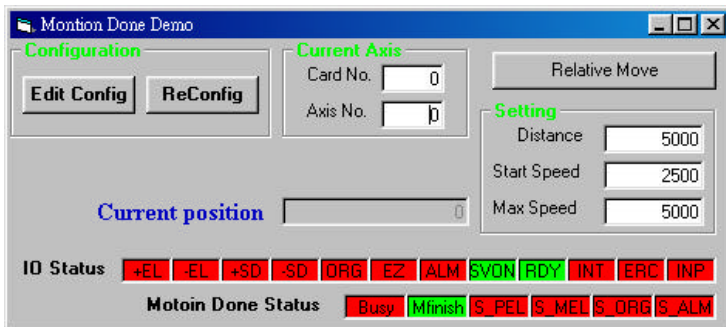
Please refer to EX2-4 directory for the complete source codes.

Example: Step by Step

- (1). Open a new project.
- (2). Add two module files: Def8134.BAS and Initial.BAS
- (3). Write Form_Load Procedure
- (4). Add the necessary Form variables and objects
- (5). Add two text boxes array to indicate these statuses
- (6). Create a timer
- (7). Read motion done status
- (8). Read I/O status
- (9). Relative motion button
- (10). Create a timer to get position
- (11). Run it

This program looks like as follows:

In Step 4) Add objects of the form



Other control items are as follows:

Object Type	Attribute	Value
Form	Name	Form1
	Caption	Simple Test
Command Button	Name	C_ShINI
	Caption	Edit Config
Command Button	Name	C_RCfg
	Caption	ReConfig
Command Button	Name	C_RMove
	Caption	Relative Move
Text Box	Name	T_Dist
	Caption	5000
Text Box	Name	T_SVel
	Caption	2500
Text Box	Name	T_MVel
	Text	500
Text Box	Name	T_CardNo
	Text	0
Text Box	Name	T_AxisNo
	Text	0
Text Box	Name	T_VStr
	Text	1000
Text Box	Name	T_VMax
	Text	5000
Text Box	Name	T_CurPos
	Text	0
Timer	Name	Timer1
	Interval	10

In Step 5) Add Text box array

Object	Name	Index
TextBox	T_IOState	0~11
TextBox	T_MotionState	0~5

In Step 6) Read Motion Done Status

3.4.2 Reading Motion Done Status

If you want to know the stopping reason of an axis, using `motion_done()` is an easy way to achieve it. The motion done status table is as follows:

Return value	Axis Status
0	Busy
1	Movement Finished
2	Stops at Positive Limit Switch
3	Stops at Negative Limit Switch
4	Stops as Origin Switch
5	Stops by Alarm Signal

The following program must be placed in timer section in order to read the status and display it constantly.

```
'Display Motion Status
MotionStatus = motion_done(Channel)
For i = 0 To 5
  If MotionStatus = i Then
    T_MotionState(i).BackColor = &HFF00&
  Else
    T_MotionState(i).BackColor = &HFF&
  End If
Next i
```

In Step 7) Read I/O Status

3.4.3 Reading I/O Status

If you want to know the I/O status of an axis, using `get_io_status()` is an easy way to do this. Each bit in I/O status value stands for one I/O status. The details of these bits are as follows:

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

In order to read I/O state and display it constantly, the following program must be placed in timer section.

```
'Display I/O Status
get_io_status Channel, IOState
For TestBit = 0 To 11
  If 2 ^ TestBit And IOState Then
    T_IOState(TestBit).BackColor = &HFF00&
  Else
    T_IOState(TestBit).BackColor = &HFF&
  End If
Next TestBit
```

In Step 8) Relative motion button

3.4.4 Relative Motion Button

There is a relative motion button in the form. When it is pressed, the axis will move for a distance. The program is as follows:

```
' Relative Button Click
Private Sub C_RMmove_Click()
  start_r_move Channel, CDb1(T_Dist.Text),
  CDb1(T_SVel.Text), CDb1(T_MVel.Text), 0.5
End Sub
```

3.5 Interrupt Handling under Windows 95/98 by Visual Basic 5.0

3.5.1 Interrupt Event

In config.exe utility, user can choose the interrupt types in the check boxes. If any one of the interrupt factors is active, the utility will automatically enable the interrupt service by set_int_control() function. If you want to use interrupts in 32bits Windows® system, you must set up an event by W_8134_Set_INT_Enable() function too. Set up an event for each axis to handle the interrupts under Windows® is the main topic in this section.

3.5.2 Create an Event

In Form_Load Procedure, user can setup an event by W_8134_INT_Enable(). Remember that every axis must has its own event handle. The program for setting events is as follows:

```
Public hEvent(4*MAX_PCI_CARDS-1) As Long

For i = 0 To TotalCard - 1
    W_8134_INT_Enable i, hEvent(4*i)
Next
```

Note: hEvent is a global array. We assign it to a maximum number of total cards for convenient. The first parameter of W_8134_INT_Enable function is card number and the second parameter is the first event address of each card.

2.5.3 Create a Thread

After setting events, you must create a thread to receive this event which is triggered by hardware interrupt. We suggest that you must create a thread and use WaitForSingleObject() WIN32 API to do this for every axis in order to get the best performance of receiving interrupts. You can also create only one thread to receive all the interrupt events and use WaitForMultipleObjects() WIN32 API to do this.

The following program tells you how to create a thread and receive interrupt events.

```
hIntThread = CreateThread(0, 0, AddressOf IntThread, 0,
    0, ThreadID)
```

There is a function name in above function. It is the thread's name. Its definition is as follows:

```
Function IntThread(ByVal Temp As Long) As Long
Dim IntCard As Long

Do
    WaitForSingleObject(hEvent(AxisNo), &HFFFFFFF)
    get_int_status AxisNo, IntStatus

    ' you can do something here

    ResetEvent (hEvent(AxisNo))
Loop While ThreadKey = True
End Function
```

Notice that if you use `WaitForSingleObject()`, you must assign the axis number in event array. There is one global variable, "ThreadKey", for control this thread. When it is true, the thread will constantly remain in PC and when it is false, the thread will end naturally.

2.5.4 Use WIN32 API in Visual Basic

In order to use WIN32 API, you must add a new module in the project to place WIN32 API declarations there.

```
Public Declare Function CreateThread Lib "kernel32"
    (ByVal lpThreadAttributes As Long, ByVal
    dwStackSize As Long, ByVal lpStartAddress As Long,
    ByVal lpParameter As Long, ByVal dwCreationFlags
    As Long, ByVal lpThreadId As Long) As Long
Public Declare Function WaitForMultipleObjects Lib
    "kernel32" (ByVal nCount As Long, lpHandles As
    Long, ByVal bWaitAll As Long, ByVal
    dwMilliseconds As Long) As Long
Public Declare Function WaitForSingleObject Lib
    "kernel32" (ByVal hHandle As Long, ByVal
    dwMilliseconds As Long) As Long
Public Declare Function CloseHandle Lib "kernel32"
    (ByVal hObject As Long) As Long
Public Declare Function ResetEvent Lib "kernel32"
    (ByVal hEvent As Long) As Long
```

2.5.5 A Complete Example

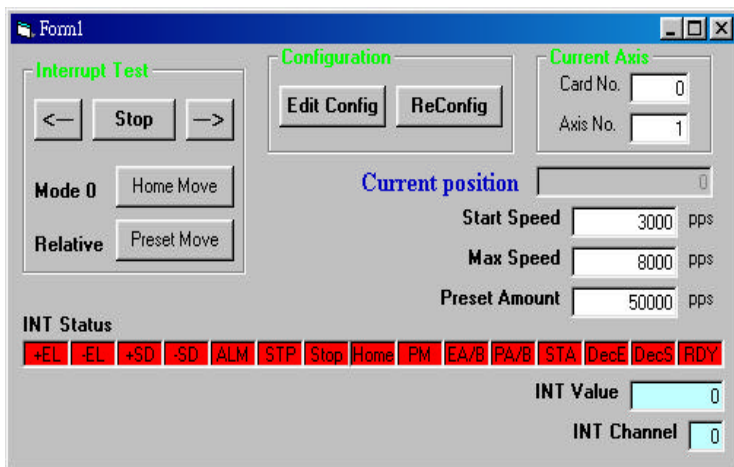
We use the same concept in last example to display interrupt status by a text box array. If interrupt comes, the thread function will receive it and decode this value. Then it will display the interrupt status on the text box array. We use `WaitForMultipleObjects()` in stead of `WaitForSingleObject()` to do this for demonstration.

In order to test interrupt signal, we design some buttons to generate it. The interrupt factors must enable properly in order to receive the signal. You must set the interrupt factor bit 5,6,13,16,17 enable because the following example will demonstrate these signals. Don't enable bit6 and bit 7 at the same time.

Example: Step by Step

- (1). Open a new project and add `Def8134.BAS` and `Initial.BAS`
- (2). Add an global module, `Global.BAS`, and place function declaration
- (3). Write `Form_Load` Procedure to initialize card and thread
- (4). Add the test buttons and objects
- (5). Decode Interrupt status
- (6). Run it

This program looks like as follows:



In Step 2) Place function declaration in Global.BAS

WIN32 APIs must be placed in this module. Please refer to 2.5.4 to insert it.

The thread is defined as follows:

```
Function IntThread(ByVal Temp As Long) As Long
    Do
        IntAxis = WaitForMultipleObjects(4 * TotalCard,
            hEvent(0), 0, &HFFFFFFF)
        get_int_status IntAxis, IntStatus

        ' you can do something here

        ResetEvent (hEvent(IntAxis))
    Loop While ThreadKey = True
End Function
```

Note: WaitForMultipleObjects() will return an axis number. It means that the axis's interrupt is triggered and the axis' corresponding event is active. In the first parameter of this function, you must put the total amount of events here. In second parameter of this function, you must put the first event address here.

In Step3) Write Form_Load Procedure to initialize card and thread

Except for Initialization and configuration, you must create events and a thread here. The program is like as follows:

```
For i = 0 To TotalCard - 1
    W_8134_INT_Enable i, hEvent(4 * i)
Next
ThreadKey = True
hIntThread = CreateThread(0, 0, AddressOf IntThread, 0,
    0, ThreadID)
```

The Boolean value, ThreadKey, can be controlled by another object in order to end this thread.

In Step 4) Add the test buttons and objects

In order to test the interrupt signal, we must use some functions to generate it. For example: Use forward or backward moving button to generate interrupt status bit 16 and bit 17. Use relative movement function to generate interrupt status bit 11. Use home function to generate interrupt status bit 9. Use stop function to generate interrupts status bit8. Please refer to example 2-5 source codes to create them.

In Step 5) Decode Interrupt Status

In order to decode interrupt status more effectively, we define a INT status array: IntArray(14). The INT status array is a continuous array from 0 to 14 but the interrupt status returned by get_int_status() is not in the same order with INT status array. So you must build a mapping table to deal it. The mapping table is as follows:

```
'INT state mapping table
IntArray(0) = 0
IntArray(1) = 1
IntArray(2) = 2
IntArray(3) = 3
IntArray(4) = 4
IntArray(5) = 5
IntArray(6) = 8
IntArray(7) = 9
IntArray(8) = 11
IntArray(9) = 12
IntArray(10) = 14
IntArray(11) = 15
IntArray(12) = 16
IntArray(13) = 17
IntArray(14) = 23
```

The left side is INT status array and the right side is actual interrupt status bits defined by get_int_status().

In order to display INT status array, you must create a corresponding text box array. The name of text box array is INTState(0 to 14). The program for decoding interrupt status and displaying on text box is as follows:

```
For TestBit = 0 To 14
  If 2 ^ IntArray(TestBit) And IntStatus Then
    Form1.INTState(TestBit).BackColor = &HFF00&
  Else
    Form1.INTState(TestBit).BackColor = &HFF&
  End If
Next
```

This program is placed in thread function.

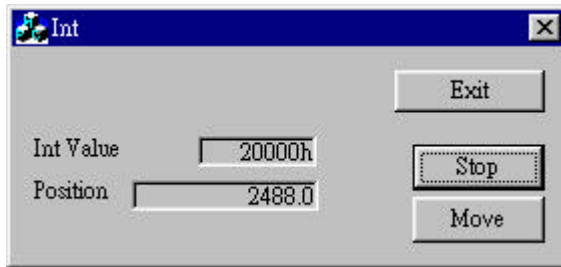
3.6 Interrupt Handling under Windows 95/98 by Visual C++ 6.0

3.6.1 Use PCI-8134 function library in Visual C++

There are two files need to be added in Visual C++ IDE if you wan to build a application under Visual C++. One is header file, pci_8134.h and the other is library file, 8134.lib. There is no configuration utility provided like VB yet so you must use function library to configure PCI-8134's parameters one by one.

3.6.2 Create a Dialog-based MFC project

We use a simple program to demonstrate interrupt function under Visual C++. First, you must create a dialog-based MFC project. The dialog resource is like as follows:



When you press Move button, axis0 will run for 10,000 pulses. During this operation, you will see the interrupt value will become 10000h when it starts. At the end if this moving, interrupt value will become 20000h and then 800h. If you press stop button, the interrupt value will become 100h.

Example 2-6: Step by Step

- (1). Create a dialog-based MFC project and include pci_8134.h and 8134.lib
- (2). Write Initial function and configure functions
- (3). Add Global variables
- (4). Create a thread in initial section
- (5). Define thread procedure in global section
- (6). Add the test buttons and objects
- (7). Create a timer and write its procedure
- (8). Run it

In Step 2) Initialize PCI-8134 and configuration

Add the following program in dialog initial section.

```
W_8134_Initial(&existCards, &pciInfo);
set_cnt_src(0,0);
set_pls_iptmode(0,2);
set_pls_outmode(0,0);
set_home_config(0,0,1,0,1);
set_inp_logic(0,1,1);
set_alm_logic(0,0,0);
set_move_ratio(0,1);
set_int_factor(0,0x032040);
W_8134_Set_INT_Control(0,1);
```

In Step 3) Add Global Variable

```
PCI_INFO pciInfo;
U16 existCards;
bool ThreadOn=false;
U32 IntState=0;
HANDLE hEvent[4];
```

In Step 4) Create Thread In Initial Section

```
W_8134_INT_Enable(0,&hEvent[0]);
ThreadOn=true;
AfxBeginThread(IntThreadProc,GetSafeHwnd(),THREAD_PRIORI
    TY_NORMAL);
```

In Step 5) Define Thread Procedure in Global Section

```
UINT IntThreadProc(LPVOID pParam)
{
    while(ThreadOn)
    {
        ::WaitForSingleObject(hEvent[0],INFINITE);

        get_int_status(0,&IntState);

        ::ResetEvent(hEvent[0]);
    }
    return true;
}
```

In Step 6) Add Move Function and Stop Function

In Stop button click event :

```
v_stop(0,0.1);
```

In Move button click event :

```
start_r_move(0,10000,100,5000,0.1);
```

In Step 7) Create a timer and write its procedure

In initial section, we create a timer :

```
SetTimer(1,100,NULL);
```

In Timer event procedure : get current command pulses and show interrupt value and position.

```
double P;  
  
get_position(0,&P);  
m_Pos.Format("%10.1f",P);  
m_IntValue.Format("%xh",IntState);  
UpdateData(false);  
CDialog::OnTimer(nIDEvent);
```

3.7 Interrupt Handling under DOS

3.7.1 A Skeleton Program for Interrupt Handling

PCI-8134 Library provides an easy way to use interrupts under DOS because it handles most of the routine works during initialization. You just need to follow the example in this section and to fill out the blank block in each procedure. The skeleton program of interrupt handling is as follows:

1) In main procedure

```
void main( void )
{
    U16  i, bn=0, axis_no=0, c_no=0;

    // ----- Initialization of PCI-8134 card -----
    _8134_Initial( &bn, &info );

    // ----- Put Other Setting for PCI-8134
    set_pls_outmode(axis_no, 0); // Pulse output mode to OUT/DIR
    set_cnt_src(axis_no, 0);     // Command as Input Counter
    set_pls_iptmode(axis_no, 2); // 4x AB phase pulse input
    set_move_ratio(axis_no, 1);  // Set Move Ration as 1
    .
    .
    .

    // ----- Set Interrupt Factor for Axis 0 -----
    set_int_factor(axis_no, 0x65); // +-EL, ALM, Home, Move

    // ----- Enable Interrupt for card 0 -----
    _8134_Set_INT_Enable( cno, 1); // Enable Int

    do {

        while(int_flag)
        {
            int_flag = 0;

            //----- You can write other codes here -----

        }
    }
```

```

} while (...)

// ----- Close all resource used by PCI-8134 -----
for(i=0; i<bn; i++)
    _8134_Close(i);
}

```

2) In ISR definition function

```

void interrupt _8134_isr0(void)
{
    U16  int_axis;
    U16  irq_status;

    disable();
    _8134_Get_IRQ_Status(0, &irq_status);
    if(irq_status)
    {
        get_int_axis(&int_axis);
        int_flag = 1;
        irq_axs = int_axis;
        get_int_status(int_axis, &irq_sts);

        // ----- You can write other codes here -----

    }
    else
        _chain_intr(pcinfo.old_isr[0]);

    outportb(0x20, 0x20);
    outportb(0xA0, 0x20);
    enable();
}

```

3.7.2 Other Empty ISR functions

No matter how many cards you use in your system, you must define 12 ISR functions in any PCI-8134 DOS program even the ISR function is empty. The function name of these 12 ISR routines must follow the styles below:

```
void interrupt _8134_isr0(void){ }
void interrupt _8134_isr1(void){ }
.
.
.
void interrupt _8134_isr9(void){ }
void interrupt _8134_isra(void){ }
void interrupt _8134_isrb(void){ }
```

A complete example for interrupt handling is in EX2-7.

3.8 Position Control by various types of Velocity Profiles

3.8.1 Velocity Profile

PCI-8134 supports 14 function types for position control in function library. The following table is the summary:

		Trapezoidal		S-Curve	
Symmetrical	Absolute	start_a_move	a_move	Start_s_move	s_move
	Relative	start_r_move	r_move	Start_rs_move	rs_move
Non-Symmetrical	Absolute	start_ta_move	ta_move	Start_tas_move	tas_move
	Relative	start_t_move	t_move	*N/A	*N/A

* It can be achieved by using absolute movement function with some tricks.

Each function has at least the following parameters:

- 1) Moving axis
- 2) Moving position or distance (depends on absolute or relative mode)
- 3) Starting Velocity
- 4) Maximum Velocity

Besides, The parameters of acceleration time depends on what velocity profile you choose. For example, a Non-Symmetrical S-Curve motion must set the following acceleration and deceleration time parameters:

- (1). Linear and S-curve acceleration time
- (2). Linear and S-curve deceleration time

3.8.2 Various types of Velocity Profile for Position Control

Example: Step by Step

- (1). Open a new project and add Def8134.BAS and Initial.BAS
- (2). Add necessary control objects and write the codes
- (3). Write Position 1 and Position 2 procedure
- (4). Run it

There are 3 control items: Current Axis, Configuration, and Current Position, which have discussed in previous examples. The "Option Button" control object must be placed in a frame container for a group.

In absolute mode, when "Position1" or "Position 2" button is pressed, the axis will move to the desired position absolutely.

In relative mode, when "Position 1" or "Position 2" button is pressed, the axis will move for a desired distance.

You can change the velocity profile setting by click the “Option Button” in “Velocity Profile” frame and “Symmetric Velocity” frame.

For each type, the corresponding parameters must be filled in the text box. You can refer to the manual for details.

This program looks like as follows:

The screenshot shows a software window titled "Velocity Profile". It contains several configuration sections:

- Action Mode:** Radio buttons for "Absolute" (selected) and "Relative".
- Velocity Profile:** Radio buttons for "T_Curve" (selected) and "S_Curve".
- Symmetric Velocity:** Radio buttons for "Yes" (selected) and "No".
- Current Axis:** Text boxes for "Card No." (0) and "Axis No." (1).
- Common Parameters:** Text boxes for "Start Velocity" (100 pps) and "Max Velocity" (25000 pps).
- Linear Acc & Dec Time:** Text boxes for "Acc. Time" (0.8 sec) and "Dec. Time" (1.2 sec).
- S_Curve Acc & Dec Time:** Text boxes for "Acc. Time" (0.6 sec) and "Dec. Time" (1.5 sec).
- Configuration:** "Edit Config" and "ReConfig" buttons.
- Current Position:** A text box showing "0".
- Stop:** A button next to "Position 1" (50000) and "Position 2" (-50000) text boxes.

In Step 1) and 2)

Please refer to the source codes in the diskette or previous examples.

In Step 3)

Write the procedure for “Position 1”

```
Private Sub C_P1_Click()  
  
    Select Case ActionMode  
        'Absolute Mode  
        Case 0
```

```

Select Case Profile
'T_Curve
Case 0
    Select Case SymOn
'Symmetric
    Case True
        start_a_move Channel, CDb1(T_P1),
CDbl(T_VStr.Text), CDb1(T_VMax.Text),
CDbl(T_Tacc.Text)
    'Non-Symmetric
    Case False
        start_ta_move Channel, CDb1(T_P1),
CDbl(T_VStr.Text), CDb1(T_VMax.Text),
CDbl(T_Tacc.Text), CDb1(T_Tdec.Text)
    End Select
'S_Curve
Case 1
    Select Case SymOn
'Symmetric
    Case True
        start_tas_move Channel, CDb1(T_P1),
CDbl(T_VStr.Text), CDb1(T_VMax.Text),
CDbl(T_Tacc.Text), CDb1(T_TSacc.Text),
CDbl(T_Tacc.Text), CDb1(T_TSacc.Text)
    'Non-Symmetric
    Case False
        start_tas_move Channel, CDb1(T_P1),
CDbl(T_VStr.Text), CDb1(T_VMax.Text),
CDbl(T_Tacc.Text), CDb1(T_TSacc.Text),
CDbl(T_Tdec.Text), CDb1(T_TSdec.Text)
    End Select
    End Select
' Relative Mode
Case 1
    Select Case Profile
'T_Curve
Case 0
    Select Case SymOn
'Symmetric
    Case True
        start_r_move Channel, CDb1(T_P1),
CDbl(T_VStr.Text), CDb1(T_VMax.Text),
CDbl(T_Tacc.Text)
    'Non-Symmetric
    Case False
        start_t_move Channel, CDb1(T_P1),
CDbl(T_VStr.Text), CDb1(T_VMax.Text),
CDbl(T_Tacc.Text), CDb1(T_Tdec.Text)
    End Select
'S_Curve
Case 1

```

```

        Select Case SymOn
        'Symmetric
        Case True
            start_rs_move Channel, CDb1(T_P1),
            CDb1(T_VStr.Text), CDb1(T_VMax.Text),
            CDb1(T_Tacc.Text), CDb1(T_TSacc.Text)
        'Non-Symmetric
        Case False
            start_tas_move Channel, CDb1(T_P1) +
            CDb1(T_CurPos.Text), CDb1(T_VStr.Text),
            CDb1(T_VMax.Text), CDb1(T_Tacc.Text),
            CDb1(T_TSacc.Text), CDb1(T_Tdec.Text),
            CDb1(T_TSdec.Text)
        End Select
    End Select
End Select

End Sub

```

Note: The Procedure of "Position 2" is similar with "Position 1". Please refer to programming guide diskette.

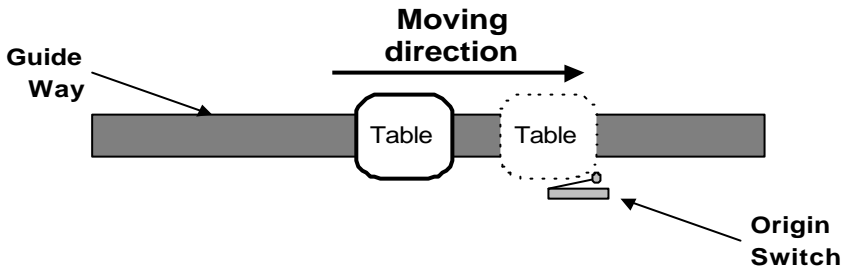
3.9 Homing routines

Home return is very important in any coordinate motion applications. Before any operations, user or program must know where is the origin point. In this section, we will show you how to find a home position in various hardware types of home input signal placement.

3.9.1 Basic Homing Styles

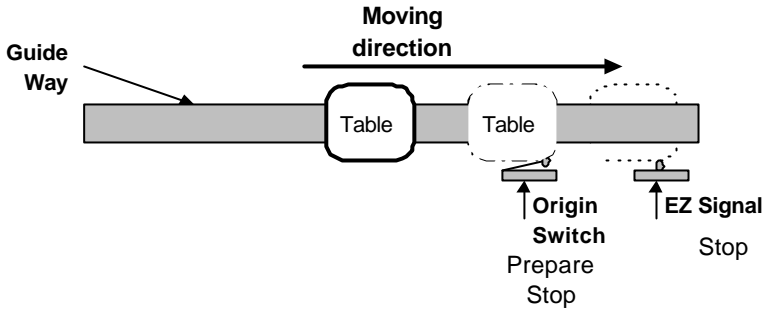
PCI-8134 function library supports 3 types of homing. It is home mode0, 1 and 2. Using which homing function depends on what types of your home input signal is arranged.

Type 0 use Home mode 0: Only one ORG input signal (ex. origin limit switch).



The Table moves to the right and touches the origin switch. Once it touches the origin switch, PCI-8134 will clear the position counter and stop the table.

Type 1 use Home mode 1: One ORG input signal and EZ input signal.

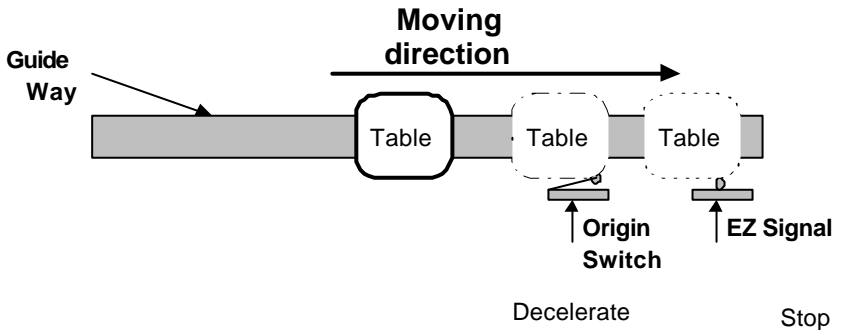


The table moves to the right and touches the origin switch. Once it touches the origin switch, PCI-8134 latches ORG signal and wait for EZ coming.

When the table moves to the EZ's trigger point, PCI-8134 will clear the position counter and stop the table.

Sometimes the EZ signal of servo motor is in its encoder. You can wire this signal without another EZ switch to PCI-8134 as EZ input.

Type 2 use mode 2: One ORG input signal and EZ input signal.



The table moves to the right and touches the origin switch. Once it touches the origin switch, PCI-8134 latches ORG signal and wait for EZ coming. At the same time, the speed of the table will decrease to the starting speed. When the table moves to the EZ's trigger point, PCI-8134 will clear the position counter and stop the table.

Notice that if your starting velocity is 0 or a very small value, the second procedure of homing will not move to the EZ position and the homing procedure will not end either.

Compare to these three methods, home mode 2 has the best accuracy on home return. But if you don't have EZ signal and you can move the table in a very slow speed when it approaches to the origin. The accuracy of homing will be guarantee too.

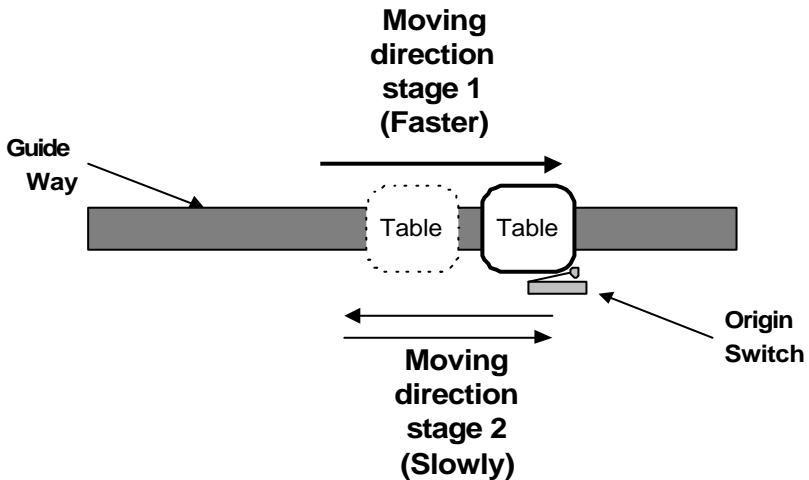
3.9.2 Advanced Homing Styles

We supply three other choices for home return in programming guide:

- (1). Two-stage homing
- (2). Midpoint Homing between positive and negative limits
- (3). Home return search

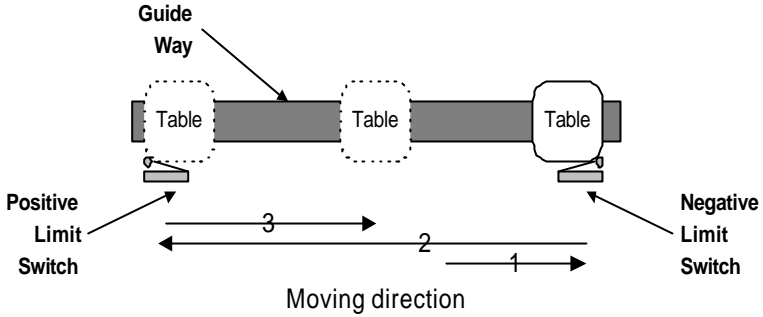
(1) Two-Stage Homing

The figure below explains what is the two stages homing. The first stage is the same with home mode 0. But it won't stop when it touches the origin switch.



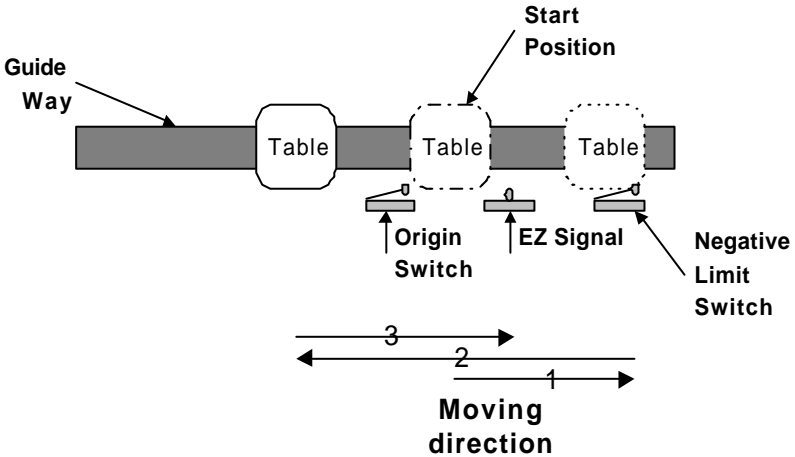
After first time it touches the origin switch, it will move off the home for a distance then move toward to the home again very slowly. When it touches the origin switch again, PCI-8134 will clear the position and stop the axis.

(2) Midpoint Homing between positive and negative limits



The figure above explains midpoint homing return. After searching the negative and positive limit switch, you can calculate the midpoint. Finally move the table to the midpoint and clear the position counter. The method can be applied to a non-origin switch system.

(3) Home return search



The figure above demonstrates the situation that the table is at between origin switch and negative limit switch. None of the basic home search in previous section will find the origin. This mode can solve this problem by move off negative limit switch for a distance and re-run the home mode search again.

3.9.3 Implement Homing Program

We have discussed three basic home return modes support by function library and three advanced home return modes in programming guide. In this section, we will demonstrate how to use these six modes in your application.

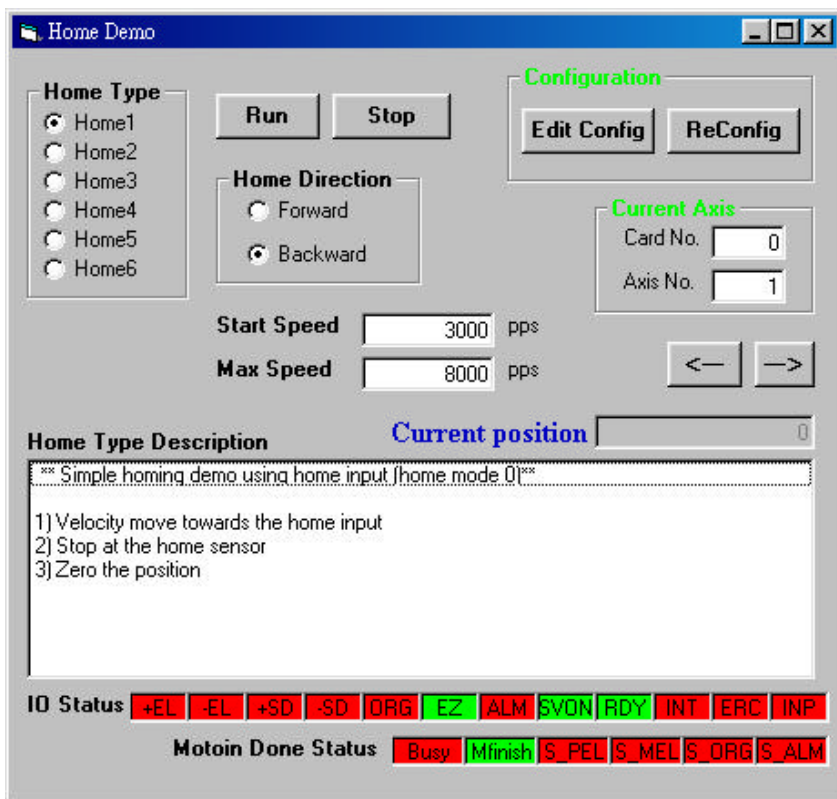
There are 6 types of home return modes in this example. Home Type 1, 3, and 4 in this program are the corresponding basic home mode 0,1 and 2. Home Type 2, 4, and 6 in this program are the corresponding advanced home mode 1,2 and 3.

Each time you click the home type button, the home type descriptions will list in the ListBox.

Example: Step by Step

- (1). Open a new project and add Def8134.BAS and Initial.BAS
- (2). Add necessary control objects and write the codes
- (3). Write Run procedure
- (4). Run it

When you press Run button, the axis will start homing according to your home type. Remember to setup your hardware before starting. This program looks like as follows:



In Step 1) and 2)

Please refer to the source codes in the diskette or previous examples.

In Step 3)

Write the procedure for “Run”

The Homing process will start when you press the Run command button. Home search direction must also be set before running.

The three basic home search method can be implement as follows:

```
Private Sub C_Run_Click()  
  
    'Select Home Type  
    Select Case Item  
        'Home Mode 0  
        Case 1  
            set_home_config Channel, 0, 1, 0, 1  
            If MoveDir = 1 Then  
                home_move Channel, CDb1(T_SVel.Text),  
                CDb1(T_MVel.Text), 0.1  
            Else  
                home_move Channel, -CDbl(T_SVel.Text), -  
                CDb1(T_MVel.Text), 0.1  
            End If  
            wait_for_done (Channel)  
            set_position Channel, 0  
        'Home Mode 1  
        Case 3  
            set_home_config Channel, 1, 1, 1, 1  
            If MoveDir = 1 Then  
                home_move Channel, CDb1(T_SVel.Text),  
                CDb1(T_MVel.Text), 0.1  
            Else  
                home_move Channel, -CDbl(T_SVel.Text), -  
                CDb1(T_MVel.Text), 0.1  
            End If  
            wait_for_done (Channel)  
            set_position Channel, 0  
            'clear latch  
            set_home_config Channel, 2, 1, 0, 1  
        'Home Mode 2  
        Case 4  
            set_home_config Channel, 2, 1, 1, 1  
            If MoveDir = 1 Then  
                home_move Channel, CDb1(T_SVel.Text),  
                CDb1(T_MVel.Text), 0.1  
            Else  
                home_move Channel, -CDbl(T_SVel.Text), -  
                CDb1(T_MVel.Text), 0.1  
            End If  
            wait_for_done (Channel)  
            set_position Channel, 0  
            'clear latch  
            set_home_config Channel, 2, 1, 0, 1  
    End Select  
  
End Sub
```

Notice that the home function's parameters (Channel, Start Velocity, Maximum Velocity) are access from the TextBox on the form. For each home type procedure, you must choose a moving direction too.

The most important thing is clearing the latch. Home mode 1 & 2 needs to latch the origin signal when the axis touches the home switch. If the latch is not cleared after home mode 1 and 2, the motion done status will become "Stop by origin" when the next movement finished. Be sure to clear the latch if you don't want it.

The three-advanced home search method can be implement as follows:

These codes are in the same section with the basic homing method.

```
Private Sub C_Run_Click()  
Dim Sta As Integer  
Dim i As Long  
Dim Pos1, Pos2 As Double  
  
i = 0  
Select Case Item  
  ' Two Stages home return  
  Case 2  
    set_home_config Channel, 0, 1, 0, 1  
    If MoveDir = 1 Then  
      home_move Channel, CDb1(T_SVel.Text),  
CDbl(T_MVel.Text), 0.1  
      wait_for_done (Channel)  
      v_move Channel, -CDbl(T_SVel.Text), -  
CDbl(T_MVel.Text), 0.1  
      Sleep 500  
      home_move Channel, 1000, 10000, 0.1  
    Else  
      home_move Channel, -CDbl(T_SVel.Text), -  
CDbl(T_MVel.Text), 0.1  
      wait_for_done (Channel)  
      v_move Channel, CDb1(T_SVel.Text),  
CDbl(T_MVel.Text), 0.1  
      Sleep 500  
      home_move Channel, -1000, -10000, 0.1  
    End If  
    wait_for_done (Channel)  
    set_position Channel, 0  
    wait_for_done (Channel) End If  
    set_position Channel, 0  
  
  'Midpoint home return  
  Case 5  
    If MoveDir = 1 Then  
      v_move Channel, CDb1(T_SVel.Text),  
CDbl(T_MVel.Text), 0.1  
    Else  
      v_move Channel, -CDbl(T_SVel.Text), -
```

```

CDBl(T_MVel.Text), 0.1
  End If
  wait_for_done (Channel)
  get_position Channel, Pos1
  If MoveDir = 1 Then
    v_move Channel, -CDBl(T_SVel.Text), -
CDBl(T_MVel.Text), 0.1
  Else
    v_move Channel, CDBl(T_SVel.Text),
CDBl(T_MVel.Text), 0.1
  End If
  wait_for_done (Channel)
  get_position Channel, Pos2
  ' Get Mid Point
  Pos2 = (Pos2 - Pos1) / 2
  set_position Channel, Pos2
  start_a_move Channel, 0, CDBl(T_SVel.Text),
CDBl(T_MVel.Text), 0.1
  wait_for_done (Channel)
  set_position Channel, 0

'Auto Search home return
Case 6
  set_home_config Channel, 2, 1, 1, 1
  If MoveDir = 1 Then
    home_move Channel, CDBl(T_SVel.Text),
CDBl(T_MVel.Text), 0.1
  Else
    home_move Channel, -CDBl(T_SVel.Text), -
CDBl(T_MVel.Text), 0.1
  End If
  wait_for_done (Channel)
  Sta = motion_done(Channel)
  'Sta=3 means stop by negative limit switch
  If Sta = 3 Then
    'clear latch
    set_home_config Channel, 2, 1, 0, 1
    start_r_move Channel, 80000,
CDBl(T_SVel.Text), CDBl(T_MVel.Text), 0.1
    wait_for_done (Channel)
    set_home_config Channel, 2, 1, 1, 1
    home_move Channel, -CDBl(T_SVel.Text), -
CDBl(T_MVel.Text), 0.1
    wait_for_done (Channel)
  End If
  set_position Channel, 0
  'clear latch
  set_home_config Channel, 2, 1, 0, 1
End Select

End Sub

```

3.10 Multiple Axes Synchronized Motion

Multiple axes synchronized motion is different from linear interpolation motion. The multiple axes motion is point to point motion so it doesn't guarantee if all the axes arrive at the same time but it guarantees they will start to move at the same time. Here is the example for this function.

3.10.1 How to use start_move_all()

The definition of start_move_all() is similar with start_a_move(). Each parameter in start_move_all() is an array. It represents the corresponding axis' motion parameters. Like axis number, position, start velocity, maximum velocity and acceleration time. You must form these parameters in an array according to its axis. For example, we have two axes for synchronized motion. Their motion parameters are as follows:

	Postion	Start velocity	Max. velocity	Acc. Time
Axis 0	150,000	1,000	25,000	0.1
Axis 3	200,000	200	30,000	0.2

We define 5 arrays to store these parameters:

```
AxisNo(0)=0,      AxisNo(1)=3
Pos(0)=150,000   Pos(1)=200,000
Svel(0)=1,000    Svel(1)=200
Mvel(0)=25,000   Mvel(1)=30,000
Tacc(0)=0.1      Tacc(1)=0.2
```

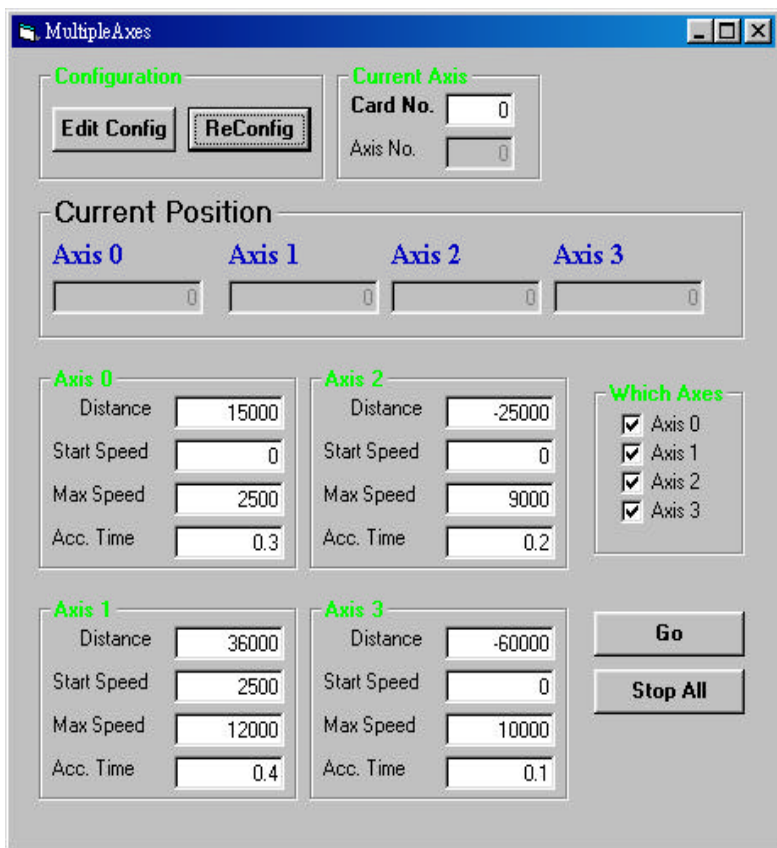
The command line is as follows:

```
start_move_all(2,AxisNo(0),Pos(0),Svel(0),Mvel(0),Tacc(0))
```

3.10.2 Implement Multiple Axes Synchronized Motion

Example: Step by Step

- (1). Open a new project and add Def8134.BAS and Initial.BAS
- (2). Add necessary control objects and write the codes
- (3). The start_move_all() parameters array
- (4). Deal the check box selection
- (5). Write Go procedure
- (6). Run it



In this example, we show one card, 4 axes synchronized motion. If you wish to use this function between different cards, you must refer to user manual of PCI-8134 to cascade the corresponding pins of CN4.

Notice that `start_move_all()` function uses absolute position value. In this example we make some change for it to move in relative position.

After you entering the corresponding parameters of every axis and choose synchronizing motion axes. Just press Go button and the axes you choose will go for a distance then stop.

In Step 1) and 2)

Please refer to the source codes in the diskette or previous examples.

In Step 3) The Start_Move_All() parameters array

```
Dim AxisMap(3) As Integer
Dim PosMap(3) As Double
Dim SVelMap(3) As Double
Dim MVelMap(3) As Double
Dim TaccMap(3) As Double
```

In Step 4) Deal with check box selection

```
For i = 0 To 3
  get_position 4 * CardNo + i, CurPos(i)
  AxisMap(i) = i
  'If the relative axis number is checked, fill it
  into a distance or let it stay in original place
  If Check(i).Value = 1 Then
    PosMap(i) = CDb1(T_Dist(i).Text) + CurPos(i)
    SVelMap(i) = CDb1(T_SVel(i).Text)
    MVelMap(i) = CDb1(T_MVel(i).Text)
    TaccMap(i) = CDb1(T_Tacc(i).Text)
  Else
    'If the relative axis number is not checked, fill
    it into a original position
    PosMap(i) = CurPos(i)
  End If
Next
```

In Step 5) Write Go Procedure

```
start_move_all 4, AxisMap(0), PosMap(0), SVelMap(0),
  MVelMap(0), TaccMap(0)
```

3.11 Linear and Circular Interpolation

We will separate this section into three parts: First and second parts give the simple codes for implementing linear and circular interpolation motion. You can add this codes into your application and show the position counter just the same with several pervious examples.

Basically, the functions `move_xy()`, `arc_xy()` map to any cards' axis0 and axis1. Also, the functions `move_zu()`, `arc_zu()` map to any cards' axis2 and axis3. Be sure that by which two axes you want to do interpolation motion before you start.

In the final part, a more complicated example is introduced here by presenting the graphics result of interpolation motion. Many WIN32 API will be applied in this example and many DC concepts will be applied too. There also has a coordinate transformation module needs to be included.

3.11.1 How to use Linear Interpolation Functions

A simple code for x-y linear interpolation is as follows

```
1) Create a two-axes mapping array
MapArray(0) = 4 * CardNo
MapArray(1) = 4 * CardNo + 1
map_axes 2, MapArray(0)

2) Set specific moving speed for the mapping array
set_move_speed StartVel, MaxVel

3) Set specific moving acceleration for the mapping array
set_move_accel Tacc

4) Start motion
move_xy CardNo, Px, Py
```

Note: In every card of interpolation motion, `move_xy()` means axis0 and axis1 ,`move_zu()` means axis2 and axis3. You can' use axis1, axis 3 or axis0, axis 2 for interpolation.

3.11.2 How to use Circular Interpolation Functions

A simple code for x-y circular interpolation is as follows:

- (1). Create a two axes mapping array

```
MapArray(0) = 4 * CardNo  
MapArray(1) = 4 * CardNo + 1  
map_axes 2, MapArray(0)
```

- (2). Set specific moving speed for the mapping array

```
set_move_speed StartVel, MaxVel
```

- (3). Set specific moving acceleration for the mapping array

```
set_move_accel Tacc
```

- (4). Set arc division resolution

```
set_arc_division DAxis, DivDegree
```

- (5). Set arc optimization on/off

```
arc_optimization Optimize
```

- (6). Start motion

```
arc_xy CardNo, CenterPx, CenterPy, MoveDegree
```

Note: In every card of interpolation motion, arc_xy() means axis0 and axis1 ,arc_zu() means axis2 and axis3. You can't use axis1, axis 3 or axis0, axis 2 for interpolation.

3.11.3 Coordinate System in Microsoft Windows®

Remember that the interpolation command we set is position counter or pulses. The scale of position counter coordinate is different from the scale of computer's coordinate system. In Windows system, the coordinate in scale mode of pixel is as follows: (ex. in a picture box control object)

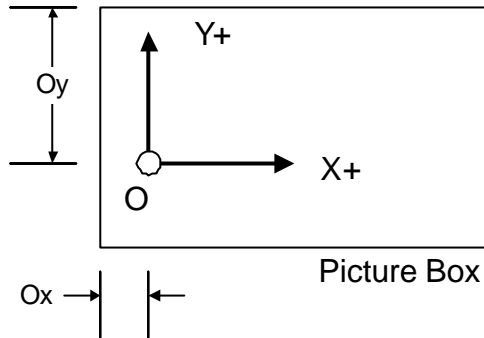
The scale unit of this picture box is pixel. In the real position counter coordinate system, the scale unit is in counter pulse. Between these two unit systems, we define a scale number:

```
'Pulses(Position counters) per pixel  
XScale = 125  
YScale = 125
```

It represents that there are 125 pulses for each pixel on the picture box,

Coordinate Transformation

The coordinate system of the position counter in picture box is as follows:



The origin point relative to pixel mode picture box coordinate system is (Ox, Oy). In the example, we set this as follows:

```
Type POINTAPI
  Px As Long
  Py As Long
End Type

Dim POrg As POINTAPI

POrg.Px = Int(Scope.ScaleWidth / 10)
POrg.Py = Int(Scope.ScaleHeight * 2 / 3)
```

The scope is the name of picture box.

Define an position counter point type

```
Type POINTREAL
  Px As Double
  Py As Double
End Type
Public PReal As POINTREAL
```

PReal stands for position counter point.

If we want to show position counter on the picture box in the position counter coordinate, we must use the function as follows to transfer a real position to picture box screen position.

```

Function Real2Scr(Pnt As POINTREAL, POrg As POINTAPI,
    ByVal XScale As Double, ByVal YScale As Double) As
    POINTAPI
    Real2Scr.Px = Pnt.Px / XScale + Abs(POrg.Px)
    Real2Scr.Py = Abs(POrg.Py) - Pnt.Py / YScale
End Function

```

Also, transfer a picture box position to a real position

```

Function Scr2Real(Pnt As POINTAPI, POrg As POINTAPI,
    ByVal XScale As Double, ByVal YScale As Double) As
    POINTREAL
    Scr2Real.Px = Int((Pnt.Px - Abs(POrg.Px)) * XScale)
    Scr2Real.Py = Int((Abs(POrg.Py) - Pnt.Py) * YScale)
End Function

```

These two functions are in Scope.BAS in Appendix D

Draw a 2-D coordinate on picture box

Further, drawing a coordinate on the picture box

```

' hDC is a destivative drawing DC
' POrg is the real system's origin relative to picture
  box
' Width is picture box's scalewidth
' Height is picture box's scaleheight
Sub PlotScale(ByVal hDC As Long, POrg As POINTAPI, ByVal
    XScale As Double, ByVal YScale As Double, ByVal
    Width As Long, ByVal Height As Long)
Dim Po As POINTAPI
Dim Incr As Long

    'Draw two cross line
    MoveToEx hDC, 0, POrg.Py, Po
    LineTo hDC, Width, POrg.Py
    MoveToEx hDC, POrg.Px, 0, Po
    LineTo hDC, POrg.Px, Height

    'X+ main scale
    Incr = Abs(POrg.Px)
    Do
        Incr = Incr + XScale
        MoveToEx hDC, Incr, POrg.Py - 2, Po
        LineTo hDC, Incr, POrg.Py + 2
    Loop While Incr < Width
    'X- main scale
    Incr = Abs(POrg.Px)
    Do
        Incr = Incr - XScale

```

```

        MoveToEx hDC, Incr, POrg.Py - 2, Po
        LineTo hDC, Incr, POrg.Py + 2
    Loop While Incr > 0

'Y- main scale
Incr = Abs(POrg.Py)
Do
    Incr = Incr + YScale
    MoveToEx hDC, POrg.Px - 2, Incr, Po
    LineTo hDC, POrg.Px + 2, Incr
Loop While Incr < Height

'Y+ main scale
Incr = Abs(POrg.Py)
Do
    Incr = Incr - YScale
    MoveToEx hDC, POrg.Px - 2, Incr, Po
    LineTo hDC, POrg.Px + 2, Incr
Loop While Incr > 0

MoveToEx hDC, POrg.Px, POrg.Py, Po

End Sub

```

This function is in Scope.BAS in Appendix D

3.11.4 DC (Device Context)

DC(Device Context) is the basic drawing object in Windows system. Besides, you must create a Bitmap object in DC before use any drawing function. You can image DC as a drawing paper. This procedure is expressed as follows:

```

Dim hScopeDC, hScopeBMP As Long

'Scope Initial
hScopeDC = CreateCompatibleDC(Scope.hDC)
hScopeBMP = CreateCompatibleBitmap(Scope.hDC,
    Scope.ScaleWidth, Scope.ScaleHeight)
SelectObject hScopeDC, hScopeBMP

```

By now, you can use hScopeDC as your drawing paper and do any drawing function like this:

```

LineTo hScopeDC, Px, Py

```

You must delete these object before leave the program (notice the delete order)

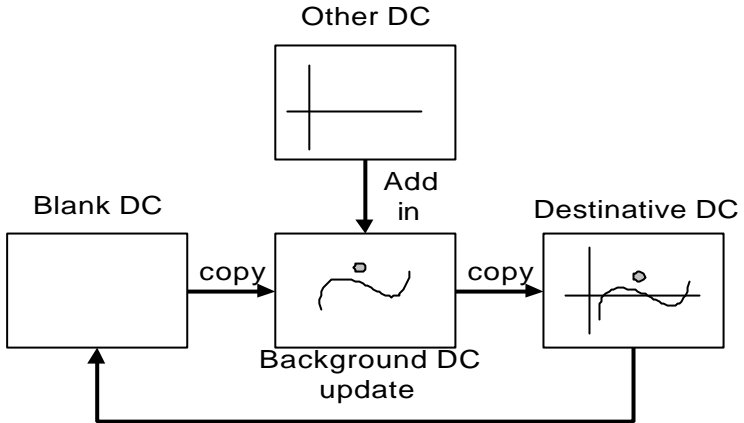
```

DeleteObject hScopeDC
DeleteObject hScopeBMP

```

3.11.5 Animation

In order to achieve a motion graph, sometimes making an animation mechanism is an important thing. This concept is expressed as follows:



Create DCs for animation

This drawing loop in previous section is done in Timer procedure. To complete this procedure, we must create at least 3 DCs as follows:

(1) Scope DC (Destinative DC)

```
hScopeDC = CreateCompatibleDC(Scope.hDC)
hScopeBMP = CreateCompatibleBitmap(Scope.hDC,
    Scope.ScaleWidth, Scope.ScaleHeight)
SelectObject hScopeDC, hScopeBMP
```

(2) Path DC (Background DC)

```
hLayerDC = CreateCompatibleDC(Scope.hDC)
hLayerBMP = CreateCompatibleBitmap(Scope.hDC,
    Scope.ScaleWidth, Scope.ScaleHeight)
SelectObject hLayerDC, hLayerBMP
```

(3) Blank DC (As Eraser)

This Blank DC must be created in picture box Paint Procedure:

```
hScopeBackDC = CreateCompatibleDC(Scope.hDC)
hScopeBackBMP = CreateCompatibleBitmap(Scope.hDC,
    Scope.ScaleWidth, Scope.ScaleHeight)
SelectObject hScopeBackDC, hScopeBackBMP
BitBlt hScopeBackDC, 0, 0, Scope.ScaleWidth,
    Scope.ScaleHeight, Scope.hDC, 0, 0, vbSrcCopy
```

Notice that there has a BitBlt function, it is very important for moving DC data to another DC. Copy or add action in DC is done by this function.

Create pens for DCs

Create two pens for DC: Black pen and White pen.

```
'Build two pen
hWhitePen = CreatePen(vbSolid, 1, RGB(255, 255, 255))
hBlackPen = CreatePen(vbSolid, 1, RGB(0, 0, 0))

'Select Pen for each DC
SelectObject hLayerDC, hBlackPen
SelectObject hScopeDC, hBlackPen
```

Select object means select an drawing tool for a DC paper.

We make a new mouse pointer for this picture box. It will show up if the mouse is moved to the region of picture box. So we must create a mouse pointer DC by loading a BMP file.

```
'Build CrossSign Cursor and its DC
Set Cross = LoadPicture(App.Path & "\" &
    "cursor.bmp")
GetObject Cross.Handle, LenB(BMP), BMP
hCursorDC = CreateCompatibleDC(Scope.hDC)
SelectObject hCursorDC, Cross.Handle
```

The BMP structure is as follows:

```
Type BITMAP
    bmType As Long
    bmWidth As Long
    bmHeight As Long
    bmWidthBytes As Long
    bmPlanes As Integer
    bmBitsPixel As Integer
    bmBits As Long
End Type
```

Animation Starts in Timer

We use timer to control the speed of animation. The codes are as follows:

```
'Clear DC
BitBlt hScopeDC, 0, 0, Scope.ScaleWidth,
    Scope.ScaleHeight, hScopeBackDC, 0, 0, vbSrcCopy

'Mark command position
Ellipse hScopeDC, PCmd.Px - 4, PCmd.Py - 4, PCmd.Px
    + 4, PCmd.Py + 4

'Set path start point on path DC & draw a line to
run time position on path DC
MoveToEx hLayerDC, PLast.Px, PLast.Py, Po
LineTo hLayerDC, PScr.Px, PScr.Py

' Paint Cross sign cursor
If CursorShow = False Then
    BitBlt hScopeDC, Cursor.Px, Cursor.Py,
        Scope.ScaleWidth, Scope.ScaleHeight, hCursorDC, 0,
        0, vbSrcCopy
End If

' Plot Scale
PlotScale hScopeDC, POrg, XDiv, YDiv,
    Scope.ScaleWidth, Scope.ScaleHeight

' Add path DC
BitBlt hScopeDC, 0, 0, Scope.ScaleWidth,
    Scope.ScaleHeight, hLayerDC, 0, 0, vbSrcAnd

' Show total drawing materials on scope
BitBlt Scope.hDC, 0, 0, Scope.ScaleWidth,
    Scope.ScaleHeight, hScopeDC, 0, 0, vbSrcCopy
```

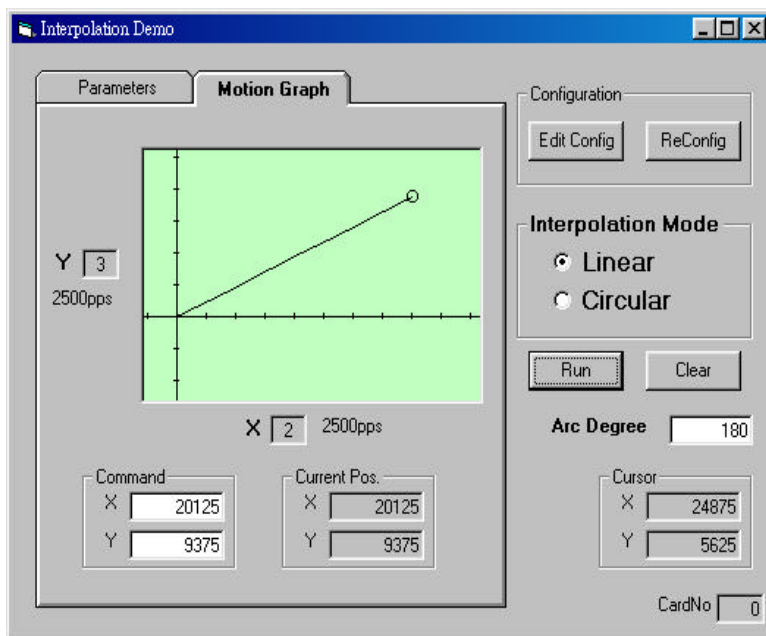
All WIN32 API declarations and self-defined functions are in Scope.Bas in Appendix D

3.11.6 Implement 2-D example

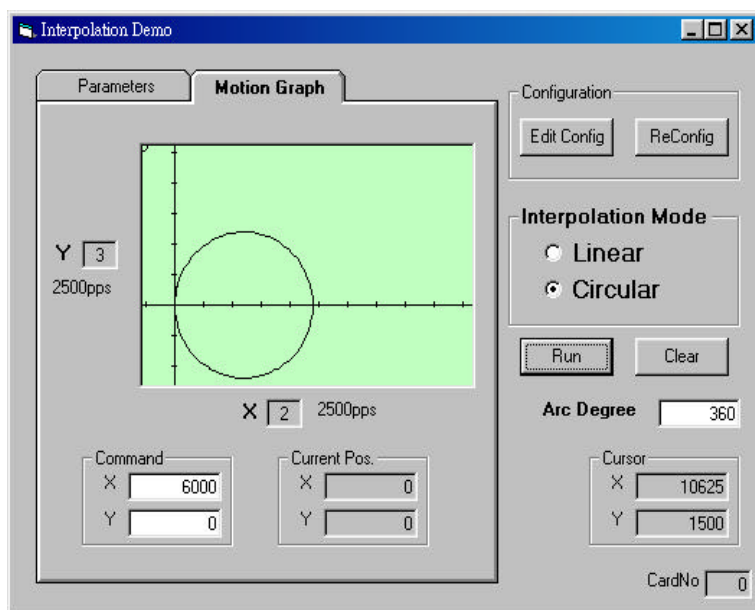
You can refer to the complete source codes or previous section for making this example. The results for this example is as follows:

Example:

(1) Linear interpolation result:



(2) The circular interpolation result



(3) Parameters Setting Page

Interpolation Demo

Parameters Motion Graph

Basic Motion Parameters

Start Speed (pps)

Max Speed (pps)

Acc Time (sec)

Circular Motion Parameters

Division Axis

Arc Division (degree)

Optimize

Configuration

Edit Config ReConfig

Interpolation Mode

Linear

Circular

Run Clear

Arc Degree

Cursor

X

Y

CardNo

3.12 Jog

The Jogging feature can be achieved by `r_move` function. Generally, there are two types of jogging. One is incremental jogging and the other is continuous jogging.

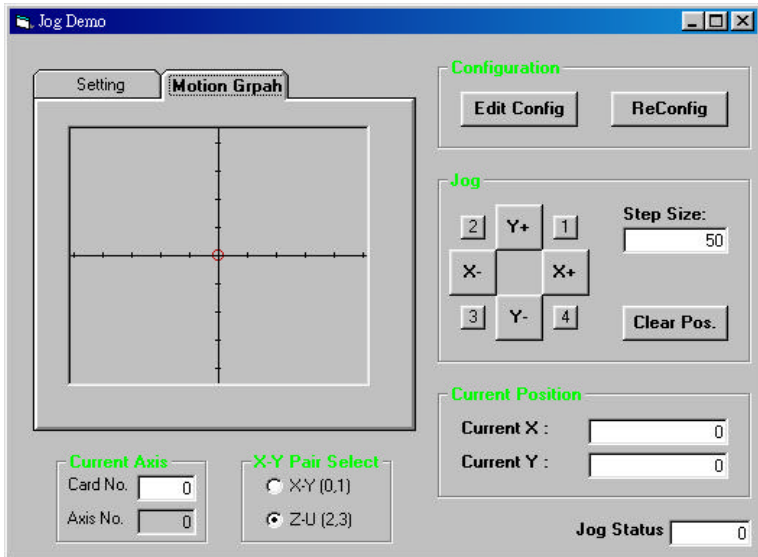
Incremental jogging means that for every times you press the jogging button, the axis will step for a distance. Continuous jogging means that when you press the jogging button and don't release it, the axis will move continuously and speed up increasingly depends on how long you press.

We combine these two types of jogging in one program. The speed increasing is proportional to the time your press the button. This feature is also can achieve by software. The changeable speed function is controlled by a seed number `i`. The function is defined as follows:

```
JogMaxVel * i ^ 2  
i=i+1 for each step
```

Example:

There are eight button in this example for 8 directions in 2-D coordinate system. This program is look like as follows:



3.12.1 Create a Thread for Jogging

You can do jogging function without any graphical presentation. Just add the following sample codes to complete it.

The codes for press the X+ direction buttons are as follows:

```
Private Sub C_XB_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    If hJogThread <> 0 Then CloseHandle (hJogThread)
    Jog = 7
    hJogThread = CreateThread(0, 0, AddressOf JogThread, 0, 0, ThreadID)
End Sub
```

Please refer to section 2.5 for creating a thread.

The codes for unpress X+ direction buttons are as follows:

```
Private Sub C_XF_MouseUP(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Jog = 0
End Sub
```

Notice that the Jog variable can tell the thread which button is pressed. Jog=0 means no button is pressed.

Because interpolation functions are only used in absolute mode, we use some tricks by increamental method to achieve the relative motion.

Please refer to example 2-12 for complete codes. Some part of thread for Jog Thread are as follows:

```
' Interpolation jogging
  Select Case Jog
    Case 1
      If Pair = 0 Then
        move_xy CardNo, LastX + Step, LastY +
Step
      Else
        move_zu CardNo, LastX + Step, LastY +
Step
      End If
      LastX = LastX + Step
      LastY = LastY + Step
    .
    .
    .
```

```

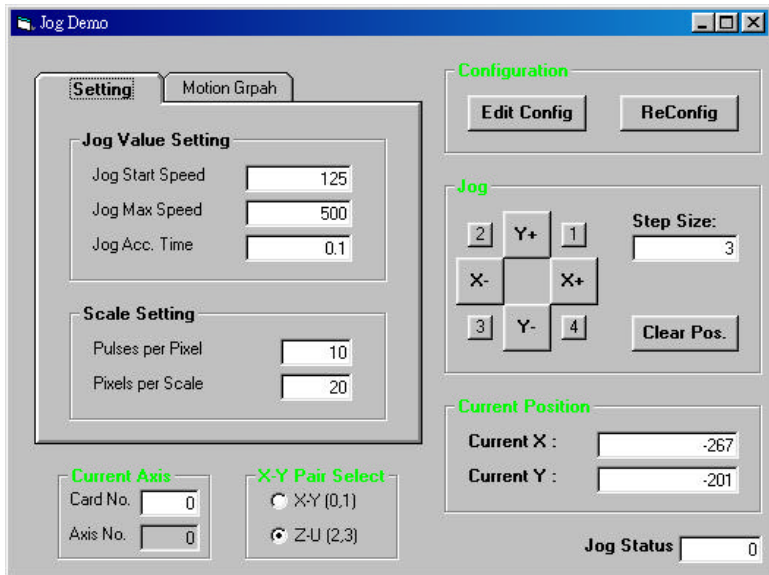
' Single axis jogging

Select Case Jog
  Case 5
    r_move 4 * CardNo + 2 * Pair, Step, JogSvel,
    JogMVel * i ^ 2, JogTacc
  Case 6
    r_move 4 * CardNo + 2 * Pair + 1, Step,
    JogSvel, JogMVel * i ^ 2, JogTacc
  Case 7
    r_move 4 * CardNo + 2 * Pair, -Step, JogSvel,
    JogMVel * i ^ 2, JogTacc
  Case 8
    r_move 4 * CardNo + 2 * Pair + 1, -Step,
    JogSvel, JogMVel * i ^ 2, JogTacc
End Select

'Increase Speed Seed
If i < 100 Then i = i + 1
'add this delay will raise the performance
Sleep (100)

```

The Setting Page for jogging is as follows:



3.12.2 Create a Scope for Display Jogging

In order to make a dynamic scope for displaying jog motion, you must create a DC for drawing. In the module Scope.BAS, there are some several functions for doing this. You can combine these three functions – CreateCompatibleDC(), CreateCompatibleBitmap(), SelectObject() into one function. For example: Creating and BMP DC, you must write:

```
hScopeDC = CreateBmpDC(Scope.hDC, Scope.ScaleWidth,  
    Scope.ScaleHeight, hScopeBmp)  
In stead of
```

```
'Scope DC (Destinative DC)  
hScopeDC = CreateCompatibleDC(Scope.hDC)  
hScopeBMP = CreateCompatibleBitmap(Scope.hDC,  
    Scope.ScaleWidth, Scope.ScaleHeight)  
SelectObject hScopeDC, hScopeBMP
```

The source codes for Scope.BAS are in Appendix D.

The following figure shows the result of jogging. Notice that The step size is 3.

3.13 Velocity Change On The Fly

3.13.1 Velocity Change on the fly

There is a v_change function in PCI-8134 library. It makes the axis change speed during continuous motion or position control motion. User can easily change speed value by pressing a Velocity Change command button during motion. The codes are as follows:

```
v_change Channel, CDb1(T_Vchg.Text), CDb1(T_Tacc.Text)
```

It also can changes the speed at the desired position or condition. The codes are as follows:

```
If Position > VChagnePos Then  
    v_change Channel, VChageValue, AccTime  
End If
```

3.13.2 Velocity Value

In order to see the velocity value without any tachometer, we use an simple way to get the velocity value by ignore the time factor. That's Measured Velocity=Current Position – LastPosition

This instruction is placed in Timer procedure. The Timer in Windows System is not accurate, so we ignore it. Besides, by using this method, there must be a high frequency noise. The results may look not very good. But at least, it is the most direct way to measure it. Or you can get the velocity by another clock or use analog input to read tachometers.

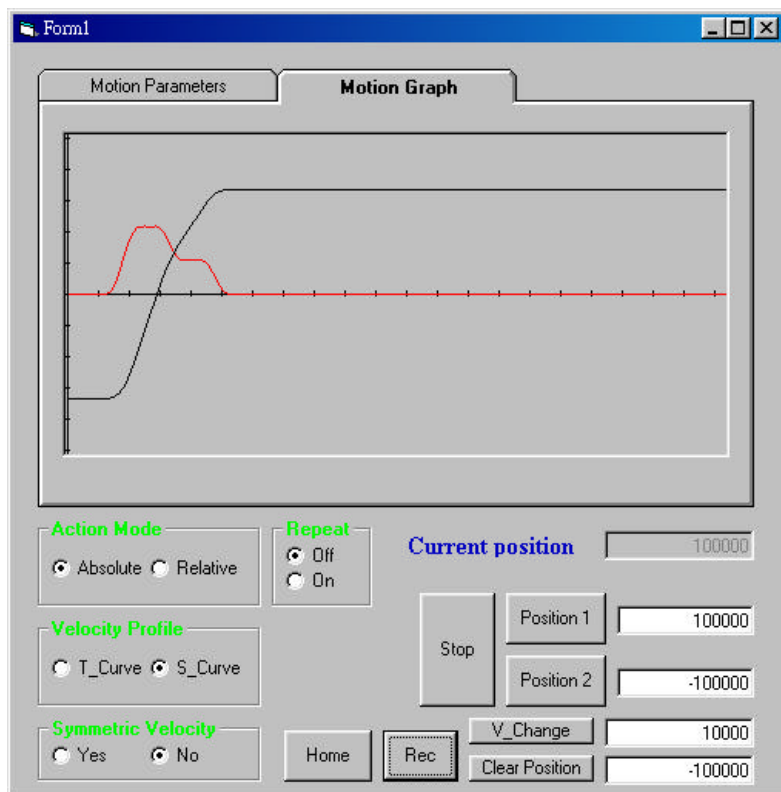
3.13.3 Velocity Change on the Fly Demo results

We test this function by setting a non-symmetric s_curve absolute mode. The demo procedure as follows:

- (1). Set the position to –100000
- (2). Start go to position 1 (100000)
- (3). Press V_Change command button during the axis is moving
- (4). Finally, the axis stops at exactly 100000

Example:

You can see the position profile (in black line) and velocity profile (in red line) in the figure.



The parameters setting are as follows:

The screenshot shows a software window titled "Form1" with a "Motion Parameters" tab. The interface is organized into several sections:

- Current Axis:** Card No. (0), Axis No. (0)
- Configuration:** Edit Config, ReConfig buttons
- Y Scale Setting:** Pulses per Pixel (1500), Pixels per Scale (20)
- Common Parameters:** Start Velocity (0 pps), Max Velocity (20000 pps)
- Linear Acc & Dec Time:** Acc. Time (1 sec), Dec. Time (2 sec)
- S_Curve Acc & Dec Time:** Acc. Time (2 sec), Dec. Time (3 sec)
- Action Mode:** Absolute (selected), Relative
- Repeat:** Off (selected), On
- Velocity Profile:** T_Curve, S_Curve (selected)
- Symmetric Velocity:** Yes, No (selected)
- Current position:** 100000
- Control Buttons:** Stop, Home, Rec, V_Change (10000), Clear Position (-100000)
- Position Inputs:** Position 1 (100000), Position 2 (-100000)

This program can test many other velocity combinations by clicking the option button in the form. Because the velocity value is getting from difference value, it may has some noise on it. For better display, you can tune the timer interval to improve it.

2.13.4 Limitation of Velocity Change on the Fly

We strongly recommend that the function `fix_max_speed()` must be set before every move function if you want to do velocity change during this moving command.

Due to the hardware limitation, user can't change axis' speed unlimitly in every moving operation. You must give PCI-8134 a maximum speed that the axis may reach first.

For example, assume that you start a moving function by setting a maximum speed of 100,000 pps and you want to change its speed to 200,000 pps when it reach some point. At this example, you must use `fix_max_speed()` to set the speed higher than 200,000 before `start_a_move()`. The codes are as follows:

```
fix_max_speed( 0, 250000);  
start_a_move( 0, 100000, 100, 100000, 0.1);
```

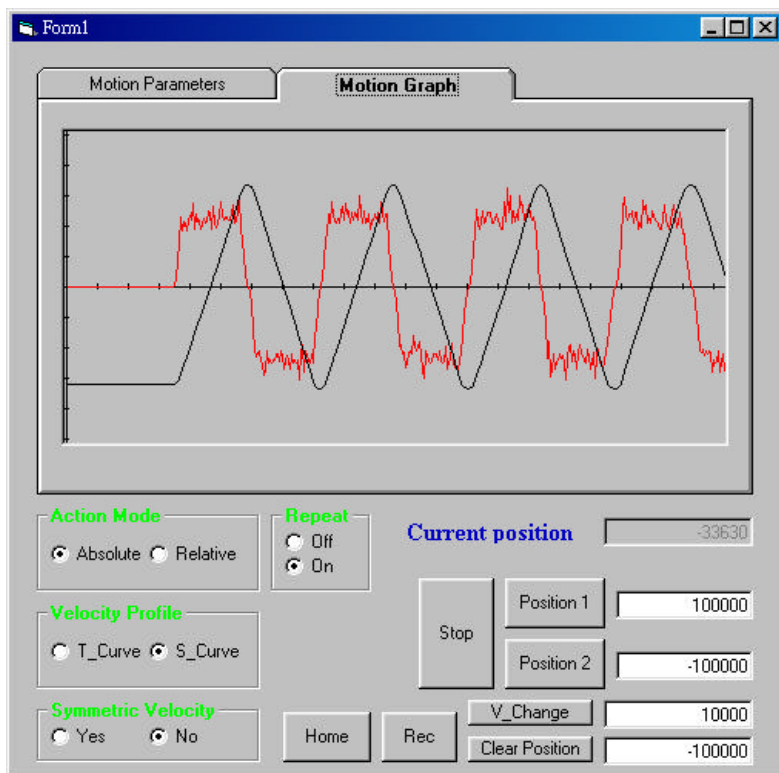
when the axis reaches 50000, change the speed to 200000

```
if( Position > 50000) v_change(0,200000,0.1);
```

3.13.5 Repeat Mode

Some applications need to run the axis cyclicly between a desired position. Making a repeat motion is necessary to introduce. The repeat motion in this program is implemented in a thread so the axis can be stopped at any moment.

The results of this motion is in the following figure.



The following program is a part of repeat mode. Please refer to EX2-13 for complete codes.

```

'Button Position 1 Pressed
Case 1

Select Case ActionMode
'Absolute mode
Case 0
Select Case Profile
'T_Curve
Case 0
' Enter the repeat motion loop
Do While RepeatOn = True
If RepeatSW = True Then
a_move Channel, P1, SVel, MVel, Tacc
RepeatSW = False
Else
a_move Channel, P2, SVel, MVel, Tacc
RepeatSW = True
End If
Loop
'S_Curve
Case 1
' Enter the repeat motion loop
Do While RepeatOn = True
If RepeatSW = True Then
s_move Channel, P1, SVel, MVel, Tacc,
Tacc
RepeatSW = False
Else
s_move Channel, P2, SVel, MVel, Tacc,
Tacc
RepeatSW = True
End If
Loop
End Select
.
.
.

```

Repeat motion loop is control by a Boolean value, RepeatOn. And it is controlled by stop button.

```

Private Sub C_Stop_Click()
v_stop Channel, CDb1(T_Tacc.Text)
RepeatOn = False
End Sub

```

4

The ISaGRAF Library for PCI-8134

4.1 Installation of PCI-8134 ISaGRAF Library

4.1.1 PCI-8134 ISaGRAF Library Installation

The Setup program provided by PCI-8134 ISaGRAF library performs all tasks necessary for installing the software.

With ADLink's "PCI-8134 ISaGRAF Library Disk" diskette :

- step 1.** Place the "PCI-8134 ISaGRAF Library" diskette in the 3.5" floppy drive A:.
- step 2.** If Windows NT is loaded, choose Run from the taskbar.
- step 3.** Type A:\SETUP in the Run dialog box.
- step 4.** When the software component installation process is complete, user has to copy the "8134isg.dll" and "ISaUSP.dll" files from the "A:\LIB" directory to "x:\ Isawin\Target\cmds" directory. (x:\Isawin indicates the directory which install the ISaGRAF software).

With "ADLink All-in-one Compact Disc":

- step 1.** Place "ADLink All-in-one Compact Disc" in the CD-ROM drive.
- step 2.** If autorun setup program is not invoked, execute x:\setup.exe(x indicates the CD-ROM drive).
- step 3.** Select Software Package-> PCI-8134 ISaGRAF Library to install the software.
- step 4.** When the software component installation process is complete, user has to copy the "8134isg.dll" and "ISaUSP.dll" files from the "x:\Software\PCI-8134 ISaGRAF Library (x indicates the CD-ROM drive)." directory to "y:\ Isawin\Target\cmds" directory. (y:\Isawin indicates the directory which install the ISaGRAF software).

4.1.2 PCI-8134 ISaGRAF Library Un-installation

PCI-8134 ISaGRAF Library has the capability of automatic un- installation. To un-install PCI-8134 ISaGRAF Library, open the “Control Panel”, double-click “Add/Remove Programs”, select “PCI-8134 ISaGRAF Library” to un-install it.

4.2 Restore PCI-8134 ISaGRAF Library C Function Objects in the ISaGRAF Workbench

3.2.1 With ADLink’s “PCI-8134 ISaGRAF C function Object” diskettes

- step 1.** Place the diskette “PCI-8134 ISaGRAF C Function Object” diskette in the 3.5" floppy drive A:.
- step 2.** Open the ISaGRAF Archive Manager Utility for “c functions”.
- step 3.** Change the “Archive Location” to “a:\ Function Definition” directory.
- step 4.** Click the Restore button, then PCI-8134 ISaGRAF library c function objects will copy to the ISaGRAF Workbench. When the copy operation finish, user click the Close button and exit this tool.

3.2.2 With “ADLink All-In-One Compact Disc”:

- step 1.** Place “ADLink All-In-One Compact Disc” in the CD-ROM drive.
- step 2.** Open the ISaGRAF Archive Manager Utility for “c functions”.
- step 3.** Because in the “ADLink All-In-One Compact Disc”, the PCIS-ISG 8134 ISaGRAF library c function objects are located in the “Software\PCI-8134 ISaGRAF Library\ Function Definition” directory, so user have to click the “Browser” button, then assign the correct directory in the “ADLink All-In-One Compact Disc”.
- step 4.** Click the Restore button, then PCI-8134 ISaGRAF library c function objects will copy to the ISaGRAF Workbench. When the copy operation finish, user click the Close button and exit this tool.

4.3 Restore PCI-8134 ISaGRAF Sample Programs

There are only one sample programs provided in this diskette. they could help you to program your own applications by using PCI-8134 ISaGRAF sample program more easily. The brief descriptions of these programs are specified as follows:

Pci8134: The introduction about use ISaGRAF with PCI-8134 card (Using SFC and ST language)

4.3.1 With ADLink's "PCI-8134 ISaGRAF Sample Program" diskettes

- step 1.** Place the diskette "PCIS-ISG 8134 ISaGRAF Sample Program" in the 3.5" floppy drive A:.
- step 2.** Open the ISaGRAF Archive Manager Utility for Project.
- step 3.** Change the "Archive Location" to "a:\ Project" directory.
- step 4.** Select the sample program user want to use, then click the Restore button, then the project will copy to the ISaGRAF Workbench. When the copy operation finish, user click the Close button and exit this tool.

4.3.2 With "ADLink All-In-One Compact Disc":

- step 1.** Place "ADLink All-In-One Compact Disc" in the CD-ROM drive.
- step 2.** Open the ISaGRAF Archive Manager Utility for Project.
- step 3.** Because in the "ADLink All-In-One Compact Disc", the PCI-8134 ISaGRAF Sample program are located in the "Software\PCI-8134 ISaGRAF Library\Project" directory, so user have to click the "Browser" button, then assign the correct directory in the "ADLink All-In-One Compact Disc".
- step 4.** Click the Restore button, then PCI-8134 ISaGRAF sample program will copy to the ISaGRAF Workbench. When the copy operation finishes, user clicks the Close button and exits this tool.

4.4 The definition of PCIS-ISG 8134 ISaGRAF Library

There are almost 60 functions in the PCI-8134 ISaGRAF library. Use can use these functions directly in ST or FBD languages on ISaGRAF environment. In order to let user can use these functions more easy. These functions be described as below:

3.4.1 Initialization function group

Function item:

- p8134ini – Software Initialization for PCI-8134
- p8134clo – Software release resources of PCI-8134
- s_config – Configure PCI-8134 according to Motion Creator
- g_irq_ch – Get the PCI-8134 card's IRQ number
- g_addres – Get the PCI-8134 card's base address

Function description :

- p8134ini:This function is used to initialize PCI-8134 card. Every PCI-8134 card has to be initialized by this function before calling other functions.
- p8134clo:This function is used to close PCI-8134 card and release the PCI-8134 related resources(This function just suport WindowNT platform).
- s_config:This function is used to configure PCI-8134 card. All the I/O configurations and some operating modes appeared on “Axis Configuration Window” of Motion Creator will be set to PCI-8134. Click “Save Configuration” button on the “Axis Configuration Window” if you want to use this function in the application program. Click “Save Configuration” button will save all the configurations to a file call “8134.cfg”. This file will appear in the “WINDOWS\SYSTEM” directory.
- g_irq_ch:This function is used to get the PCI-8134 card's IRQ number. (This function just suport Window 95 and Window NT platform only).
- g_addres:This function is used to get the PCI-8134 card's base address.

Syntax

```
Analog result = p8134ini (Analog cardNo)
Analog result = p8134clo (Analog cardNo)
Analog result = s_config (Message file)
Analog irq_no = g_irq_ch(Analog cardNo)
Analog base_addr = g_addres (Analog cardNo)
```

Input parameter :

card_no : The PCI-8134 card index number
file : The name of PCI-8134 card configuration file created by Motion Creator

Output parameter :

base_addr : The PCI-8134 card's base address.
irq_no: The PCI-8134 card's IRQ number.
Result : 0 – No Error , >0 – Error

4.4.2 Pulse Input /Output Configuration function group

Function item :

- s_PlsOut – Set the configuration for pulse command output.
- s_PlsIpt – Set the configuration for feedback pulse input.
- s_CntSrc – Enable/Disable the external feedback pulse input

Function description :

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

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

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

Syntax

```
Analogue result = s_PlsOut (Analogue axis, Analogue  
    pls_outmode)  
Analogue result = s_PlsIpt (Analogue axis,  
    Analogue pls_iptmode)  
Analogue result = s_config (Analogue axis, Analogue  
    cnt_src)
```

Input parameter :

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

pls_outmode : setting of command pulse output mode for OUT and DIR pins.

pls_outmode=0, OUT/DIR type pulse output.

pls_outmode=1, CW/CCW type pulse output.

pls_inpmode : setting of encoder feedback pulse input mode for EA and EB pins.

pls_iptmode=0, 1X AB phase type pulse input.

pls_iptmode=1, 2X AB phase type pulse input.

pls_iptmode=2, 4X AB phase type pulse input.

pls_iptmode=3, CW/CCW type pulse input.

cnt_src : Counter source

cnt_src=0, counter source from command pulse

cnt_src=1, counter source from external input EA,

EB

Output parameter :

Result : 0 – No Error, >0 – Error

4.4.3 Continuously Motion Move function group

Function item :

- v_move – Accelerate an axis to a constant velocity with trapezoidal profile
- sv_move – Accelerate an axis to a constant velocity with S-curve profile
- v_change – Change speed on the fly
- v_stop – Decelerate to stop

Function description :

- v_move : This function is used to accelerate an axis to the specified constant velocity. The axis will continue to travel at a constant velocity until the velocity is changed or the axis is commanded to stop. The direction is determined by the sign of velocity parameter..
- sv_move: This function is similar to v_stop() but accelerating with S-curve.
- v_change : You can change the velocity profile of command pulse output during operation by this function. This function changes the maximum velocity setting during operation. However, if you operate under “Preset Mode” (like start_a_move(),...), you are not allowed to change the acceleration parameter during operation because the deceleration point is pre-determined. But changing the acceleration parameter when operating under “Constant Velocity Mode” is valid.
- v_stop : This function is used to decelerate an axis to stop. This function is also useful when preset move(both trapezoidal and S-curve motion), manual move or home return function is performed.

Syntax

```
Analogue result = v_move (Analogue axis, Real str_vel,
                          Real max_vel, Real accel)
Analogue result = sv_move (Analogue axis, Real str_vel,
                           Real max_vel, Real tlacc, Real tsacc)
Analogue result = v_change (Analogue axis, Real max_vel,
                            Real accel)
Analogue result = v_stop (Analogue axis, Real decel)
```

Input parameter :

- axis : axis number designated to move or stop.
- str_vel : starting velocity in unit of pulse per second
- max_vel : maximum velocity in unit of pulse per second
- accel, tlacc : specified acceleration time in unit of second
- decel, tsacc : specified deceleration time in unit of second

Output parameter :

- Result : 0 – No Error , >0 – Error

3.4.4 Trapezoidal Motion Mode function group

Function item:

- s_a_mov– Begin an absolute trapezoidal profile motion
- s_r_mov– Begin a relative trapezoidal profile motion
- s_t_mov– Begin a non-symmetrical relative trapezoidal profile motion
- s_ta_mov– Begin a non-symmetrical absolute trapezoidal profile motion
- a_move– Begin an absolute trapezoidal profile motion and wait for completion
- r_move– Begin a relative trapezoidal profile motion and wait for completion
- t_move– Begin a non-symmetrical relative trapezoidal profile motion and wait for completion
- ta_move– Begin a non-symmetrical absolute trapezoidal profile motion and wait for completion

Function description:

- s_a_mov : This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the specified absolute position, immediately returning control to the program. The acceleration rate is equal to the deceleration rate. a_move() starts an absolute coordinate move and waits for completion...
- s_r_mov : This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the relative distance, immediately returning control to the program. The acceleration rate is equal to the deceleration rate. r_move() starts a relative move and waits for completion.
- s_t_mov : This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the relative distance, immediately returning control to the program.. t_move() starts a relative coordinate move and waits for completion.
- s_ta_mov : This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the specified absolute position, immediately returning control to the program.. ta_move() starts an absolute coordinate move and waits for completion.

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

Syntax

```
Analog result = s_a_move (Analog axis, Real pos, Real
    str_vel, Real max_vel, Real accel)
Analog result = s_r_move (Analog axis, Real distance,
    Real str_vel, Real max_vel, Real accel)
Analog result = s_t_move (Analog axis, Real distance,
    Real str_vel, Real max_vel, Real accel, Real
    decel)
Analog result = s_ta_mov (Analog axis, Real pos, Real
    str_vel, Real max_vel, Real tacc, Real tdec)
Analog result = a_move (Analog axis, Real pos, Real
    str_vel, Real max_vel, Real accel)
Analog result = r_move (Analog axis, Real distance,
    Real str_vel, Real max_vel, Real accel)
Analog result = t_move (Analog axis, Real distance,
    Real str_vel, Real max_vel, Real accel, Real
    decel)
Analog result = ta_move (Analog axis, Real pos, Real
    str_vel, Real max_vel, Real tacc, Real tdec)
```

Input parameter :

axis : axis number designated to move or stop.

pos : specified absolute position to move

distance : specified relative distance to move

str_vel : starting velocity in unit of pulse per second

max_vel : maximum velocity in unit of pulse per second

accel, tacc : specified acceleration time in unit of second

decel, tdec : specified acceleration time in unit of second

Output parameter :

Result : 0 – No Error , >0 – Error

4.4.5 S-Curve Profile Motion function group

Function item:

- s_s_move – Begin a S-Curve profile motion
- s_move – Begin a S-Curve profile motion and wait for completion
- s_rs_move– Begin a relative S-Curve profile motion
- rs_move– Begin a relative S-Curve profile motion and wait for completion
- s_tas_mov– Begin a non-symmetrical absolute S-curve profile motion
- tas_move– Begin a non-symmetrical absolute S-curve profile motion and wait for completion

Function description :

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

Syntax

```

Analog result = s_s_move (Analog axis, Real pos, Real
    str_vel, Real max_vel, Real tlacc, Real tsacc)
Analog result = s_move (Analog axis, Real pos, Real
    str_vel, Real max_vel, Real tlacc, Real tsacc)
Analog result = s_rs_mov (Analog axis, Real distance,
    Real str_vel, Real max_vel, Real tlacc, Real
    tsacc)
Analog result = rs_move (Analog axis, Real distance,
    Real str_vel, Real max_vel, Real tlacc, Real
    tsacc)
Analog result = s_tas_mo (Analog axis, Real pos, Real
    str_vel, Real max_vel, Real tlacc, Real tsacc,
    Real tldec, Real tsdec)
Analog result = tas_move (Analog axis, Real pos, Real
```

str_vel, Real max_vel, Real tlacc, Real tsacc,
Real tldec, Real tsdec)

Input parameter :

axis : axis number designated to move.

pos : specified absolute position to move

distance : specified relative distance to move

str_vel : starting velocity in unit of pulse per second

max_vel : maximum velocity in unit of pulse per second

tlacc : specified linear acceleration time in unit of
second

tsacc : specified S-curve acceleration time in unit of
second

tldec : specified linear deceleration time in unit of
second

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

Output parameter :

Result : 0 – No Error, >0 – Error

4.4.6 Linear and Circular Interpolated Motion function group

Function item :

- Y move_xy – Perform a 2-axes linear interpolated motion between X & Y
- U move_zu – Perform a 2-axes linear interpolated motion between Z & U
- Y arc_xy – Perform a 2-axes circular interpolated motion between X & Y
- Y arc_xy – *Perform a 2-axes circular interpolated motion between Z & U*

Function description :

- move_xy : These two functions cause a linear interpolation motion between two axes and wait for completion. The moving speed should be set before performing these functions.
- Move_zu : These two functions cause a linear interpolation motion between two axes and wait for completion. The moving speed should be set before performing these functions.
- Arc_xy : These two functions cause the axes to move along a circular arc and wait for completion. The arc starts from origin and continues through the specified angle. A positive value for angle produces clockwise arcs and a negative value produces counter-clockwise arcs. The center of the arc is specified by the parameters x_center and y_center. set_arc_division() function specifies the maximum angle(in degrees) between successive points along the arc. The default angle is 5 degrees. The moving speed should be set before performing these functions.
- Arc_zu : These two functions cause the axes to move along a circular arc and wait for completion. The arc starts from origin and continues through the specified angle. A positive value for angle produces clockwise arcs and a negative value produces counter-clockwise arcs. The center of the arc is specified by the parameters x_center and y_center. set_arc_division() function specifies the maximum angle(in degrees) between successive points along the arc. The default angle is 5 degrees. The moving speed should be set before performing these functions.

Syntax

```
Analog result = move_xy (Analog card_no, Real x, Real
                        y)
Analog result = move_zu (Analog card_no, Real z, Real
                        u)
Analog result = arc_xy (Analog card_no, Real x_center,
                       Real y_center, Real angle)
Analog result = arc_zu (Analog card_no, Real z_center,
```

Real u_center, Real angle)

Input parameter :

card_no : card number designated to perform interpolating function.
x, y, z, u : absolute target position of linear interpolation motion
x_center, y_center, z_center, u_center : center position of an arc
angle : specified angle for an arc

Output parameter :

Result : 0 – No Error , >0 – Error

4.4.7 Interpolation Parameters Configuring function group

Function item :

s_m_sped – Set the vector velocity
s_m_accl – Set the vector acceleration time
s_ArcDiv – Set the interpolation arc segment length
arc_opti – Enable/Disable optimum acceleration calculations for arcs
s_mratio – Set the axis resolution ratios

Function description :

S_m_sped: The vector velocity and vector acceleration can be specified for coordinated motion by this two functions.
S_m_accl: The vector velocity and vector acceleration can be specified for coordinated motion by this two functions.
S_arcdiv: This function specifies the maximum angle (in degrees) between successive points along the arc. The default is 5 degrees..
Arc_opti: This function enables (optimize = TRUE) or disable (optimize = FALSE) the automatic calculation of the optimum acceleration for an arc. The default state for arc optimization is enabled..
S_mratio: This function configures scale factors for the specified axis. Usually, the axes only need scale factors if their mechanical resolutions are different. For example, if the resolution of feedback sensors is two times resolution of command pulse, then ratio = 2..

Syntax

```
Analog result = s_m_sped (Real str_vel, Real max_vel)
Analog result = s_m_accl (Real accel)
Analog result = s_arcdiv (Analog aixs, Real degree)
Analog result = arc_opti (Analog optimize)
Analog result = s_mratio (Analog axis, Real ratio)
```

Input parameter :

axis : axis number designated to configure
 str_vel : starting velocity in unit of pulse per second
 max_vel : maximum velocity in unit of pulse per second
 accel : specified acceleration time in unit of second
 optimize : enables (*optimize* = 1) or disable (*optimize* = 0) the automatic calculation of the optimum acceleration for an arc.
 degree : maximum angle between successive points along the arc.
 Ratio : ratio of (feedback resolution)/(command resolution)

Output parameter :

Result : 0 – No Error , >0 – Error

4.4.8 Interpolation Parameters Configuring function group**Function item :**

s_h_cofg –Set the configuration for home return.
 home_mov –Perform a home return move.

Function description :

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

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

Syntax

```

Analog result = s_h_cofg (Analog axis, Analog
                        home_mode, Analog org_logic, Analog
                        org_latch, Analog ez_logic)
Analog result = home_mov (Analog axis , Real
                        str_vel, Real max_vel, Real accel)

```

Input parameter :

axis : axis number designated to configure and perform home returning

home_mode : stopping modes for home return. home_mode=0, ORG active only. home_mode=1, ORG active and then EZ active to stop, high speed all the way. home_mode =2, ORG active and then EZ active to stop, high speed till ORG active then low speed till EZ active.

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

org_latch : Latch state control for ORG signal
org_latch=0, latch input; org_latch=1, latch input.

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

str_vel : starting velocity in unit of pulse per second

max_vel : maximum velocity in unit of pulse per second

accel : specified acceleration time in unit of second

Output parameter :

Result : 0 – No Error , >0 – Error

4.4.9 Manual Pulser Motion function group**Function item :**

s_mauipt –Set pulser input mode and operation mode.

manu_mov –Begin a manual pulser movement

Function description :

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

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

Syntax

```

Analog result = s_mauipt (Analog axis, Analog
manu_iptmode, Analog op_mode)
Analog result = manu_mov (Analog axis , Real
max_vel,)

```

Input parameter :

axis : axis number designated to start manual move

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

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

max_vel : maximum velocity in unit of pulse per second

Output parameter :

Result : 0 – No Error , >0 – Error

4.4.10 Motion Status function group**Function item :**

mot_done –Return the status when a motion is done.

Function description :

mot_done : Return the motion status of PCI-8134. position.

Definition of return value is as following:

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

Syntax

 Analog result = mot_done (Analog axis)

Input parameter :

axis : axis number designated to start manual move

Output parameter :

Result :

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

4.4.11 Servo Drive Interface function group

Function item :

- s_AlmLog* –Set alarm logic and alarm mode
- s_InpLog* –Set In-Position logic and enable/disable
- s_SdLog* –Set slow down point logic and enable/disable
- s_ErcEnl* –Set ERC pin output enable/disable

Function description :

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

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

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

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

Syntax

```
Analogue result = s_AlmLog (Analogue axis, Analogue
    alm_logic, Analogue    alm_mode)
Analogue result = s_InpLog (Analogue axis, Analogue
    inp_logic, Analogue    inp_enable)
Analogue result = s_SdLog (Analogue axis, Analogue    sd_logic,
    Analogue sd_latch, Analogue sd_enable)
Analogue result = s_ErcEnl (Analogue axis, Analogue
    erc_enable)
```

Input parameter :

axis : axis number designated to configure

alm_logic : setting of active logic for ALARM signal
alm_logic=0, active LOW.

alm_logic=1, active HIGH.

alm_mode : reacting modes when receiving ALARM signal.

alm_mode=0, motor immediately stops.

alm_mode=1, motor decelerates then stops.

inp_enable : INP function enable/disable

inp_enable=0, Disabled

inp_enable=1, Enabled

inp_logic : setting of active logic for INP signal

inp_logic=0, active LOW.

inp_logic=1, active HIGH.

sd_logic : setting of active logic for SD signal

sd_logic=0, active LOW.

```

sd_logic=1, active HIGH.
sd_enable : Slow down point function enable/disable
sd_enable=0, Disabled
sd_enable=1, Enabled
sd_latch : setting of latch control for SD signal
sd_logic=0, do not latch.
sd_logic=1, latch.
Erc_enable : ERC pin output enable/disable
erc_enable=0, Disabled
erc_enable=1, Enabled

```

Output parameter :

Result : 0 – No Error , >0 – Error

4.4.12 I/O Control and Monitoring function group

Function item :

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

Function description :

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

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

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

Syntax

```

Analog result = set_svon (Analog    axis, Analog
                        on_off)
Analog io_sts = g_iostus (Analog    axis)

```

Input parameter :

axis : axis number for I/O control and monitoring

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

Output parameter :

result : 0 – No Error , >0 – Error

io_stus : I/O status word. Where “1” is ON and “0” is OFF.

4.4.13 Position Control function group**Function item :**

set_pos –Set the actual position.

get_pos –Get the actual position.

set_comd –Set the current command position.

get_comd –Get the current command position.

Function description :

set_pos: changes the current actual position to the specified position

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

set_comd: changes the command position to the specified command position.

get_comd: reads the current command position.

Syntax

```

Analog  result = set_pos (Analog  axis, Real  spos)
Analog  gpos = get_pos (Analog  axis)
Analog  result = set_comd (Analog  axis, Real  spos)
Analog  gpos = get_comd (Analog  axis)

```

Input parameter :

axis : axis number designated to set and get position.

spos : actual position or command position

Output parameter :

Result : 0 – No Error , >0 – Error

gpos : actual position or command position

4.4.14 Interrupt Control function group

Function item :

- s_IntCor –Set interrupt control status
- s_IntFac –Set interrupt generating factors
- r_int_ax –Get the axis which generates interrupt
- r_int_st –Get the interrupting status of axis

Function description :

- s_IntCor : This function is used to enable/disable INT control
- s_IntFac : This function allows users to select factors to initiate the INT signal. Enter 1 in each bit to output INT signal to host PC according to the factor set.

The definition for each bit is as following:

Bit	Interrupt Factor
0	Stop with the EL signal
1	Stop with the SD signal
2	Stop with the ALM signal
3	Stop with the STP signal
4	x(should be set to 0)
5	Completion of home return
6	Completion of preset movement
7	Completion of interpolating motion for two axes: (X & Y) or (Z & U)
8~12	x(should be set to 0)
13	when v_stop() function stop the axis
14	EA/EB, PA/PB encoder input error
15	start with STA signal
16	Completion of Acceleration
17	Start of Deceleration
18~22	Should be 0
23	RDY active (AP3 of PCL5023 change from 1 to 0)
24~31	x(should be set to 0)

Bit 14: When pins EA and EB, or PA and PB change simultaneously, it will result in an encoder input error, with these pins made valid.

r_int_ax : This function can be used inside the Interrupting Service Routine(ISR) to identify which axis generates the INT signal to host PC.

r_int_st : This function is also used inside ISR. When knowing the interrupt requested axis by

get_int_axis(), user should read the interrupt factor by this function.

The definition of each bit for **int_status* is as following:

Bit	Interrupt Factor
0	Stop with the +EL signal
1	Stop with the -EL signal
2	Stop with the +SD signal
3	Stop with the -SD signal
4	Stop with the ALM signal
5	Stop with the STP signal
6	0
7	0
8	Stop with v_stop() command
9	Stop with home return completed
10	0
11	Stop with preset movement completed
12	Stop with EA/EB input error
13	0
14	Stop with PA/PB input error
15	Start with STA signal
16	Deceleration Completed
17	Acceleration Starting
18~22	Should be 0
23	RDY active(AP3 of PCL5023 change from 1 to 0)
24~31	0

Syntax

```

Analog result = s_IntCor (Analog card_no, Analog
                          iniflag)
Analog gpos = s_IntFac (Analog axis, Analog
                       int_factor)
Analog int_axis = r_int_ax (Analog card_no)
Analog int_status = r_int_st (Analog card_no)

```

Input parameter :

card_no : card number wanting to enable interrupt generating
axis : axis number wanting to set and read interrupt factor.
Iniflag : the init flag(1 : enable, 0 : disable)
int_factor : interrupting factor to set

Output parameter :

Result : 0 – No Error , >0 – Error
int_axis : axis number which generates interrupt
int_status : interrupt factor monitor

4.5 The mapping between PCIS-8134 NT DLL function and PCI-8134 ISaGRAF Library

PCI-8134 NT DLL function	PCIS-ISG 8134 ISaGRAF Library
W_8134_Initial	p8134ini
W_8134_Close	p8134clo
W_8134_Set_SVON	set_svon
W_8134_Get_IRQ_Status	Not implementation
W_8134_Get_IRQ_Channel	g_irq_ch
W_8134_Get_Base_Addr	g_addres
W_8134_Set_INT_Control	s_IntCor
W_8134_Set_Config	s_config
start_a_move	s_a_move
a_move	a_move
start_r_move	s_r_move
r_move	r_move
start_t_move	s_t_move
t_move	t_move
wait_for_done	Not implementation
set_move_ratio	s_MRatio
get_position	get_pos
set_position	set_pos
get_command	get_comd
set_command	set_comd
v_move	v_move
sv_move	sv_move
v_change	v_chang
v_stop	v_stop
get_io_status	g_iostus
motion_done	mot_done
map_axes	not implementation
set_move_mode	not implementation
set_move_pos	not implementation
set_move_speed	s_m_sped
set_move_accel	s_m_ace1
set_move_saccel	not implementation
start_motion	not implementation
stop_motion	not implementation
set_sync_mode	not implementation
set_arc_division	s_ArcDiv
arc_optimization	arc_opti
move_xy	move_xy
move_zu	move_zu

arc_xy	arc_xy
arc_zu	arc_zu
set_home_config	s_h_cfg
home_move	home_mov
set_manu_iptmode	s_Maulpt
manu_move	manu_mov
set_pls_outmode	s_PlsOut
set_pls_iptmode	s_PlsIpt
set_cnt_src	s_CntSrc
set_alm_logic	s_AlmLog
set_inp_logic	s_InpLog
set_erc_enable	s_ErcEnl
set_sd_logic	s_SdLog
set_int_factor	s_IntFac
Read_Int_Axis	r_int_ax
Read_Int_Status	r_ints_st
W_8134_INT_Enable	not implementation
W_8134_INT_Disable	not implementation
start_ta_move	s_ta_mov
ta_move	ta_move
start_s_move	s_s_move
s_move	s_move
start_rs_move	s_rs_mov
rs_move	rs_move
start_tas_move	s_tas_mov
tas_move	tas_move
start_move_all	not implementation
move_all	not implementation
wait_for_all	not implementation