

ADT-8940A1

4-axis Motion Control Card User's Reference Manual

28 August 2009

Rev 4.0



Adtech (Shenzhen) CNC Technology Co., Ltd

Add: Building 36 Majialong Industrial Area Nanshan District, Shenzhen
Post Code: 518052

Tel: +86 755 2609 9116

Fax: +86 755 2609 2719

E-mail: export@adtechen.com

Technical support: support@adtechen.com

Copyrights

This User Manual contains proprietary information held by Adtech (Shenzhen) CNC Technology Co., Ltd (“Adtech” hereafter); stimulation, copy, photocopy or translation into other languages to this User Manual shall be disallowed unless otherwise approved by Adtech.

Meanwhile, Adtech doesn't provide any kind of warranty, expression on standing or implication. Adtech and its staffs are not liable for any direct/ indirect information disclosure, economic loss or progress termination caused by this User Manual and the product information inside.

All the contents in this User Manual may be changed without any notice.

Trademark

All the product names introduced in this User Manual are only for identification purpose, while they may belong to other various trademarks or copyrights, such as:

- ※ INTEL and PENTIUM are trademarks of INTEL Company;
- ※ WINDOWS and MS-DOS trademarks of MICROSOFT Company;
- ※ ADT-8940 is the trademark of Adtech;
- ※ Other trademarks belong to their corresponding registered companies.

All copyrights reserved by Adtech (Shenzhen) CNC Technology Co., Ltd

Version Upgrading Record

Version	Revised in	Descriptions
V4.0	2009/08/28	The fourth version

Remark: The three digits in the version number respectively mean:



Hardware version number



Major version number



Minor version number

Contents

CHAPTER 1 GENERAL INFORMATION	- 6 -
☞ INTRODUCTION	- 6 -
☞ MAIN FUNCTIONS	- 6 -
☞ APPLICATION SCOPE	- 7 -
CHAPTER 2 HARDWARE INSTALLATION	- 8 -
☞ PARTS	- 8 -
☞ INSTALLATION	- 8 -
CHAPTER 3 ELECTRICAL CONNECTION.....	- 9 -
☞ J1 LINE	- 9 -
☞ J2 LINE	- 13 -
☞ CONNECTION FOR PULSE/ DIRECTION INPUT SIGNAL	- 16 -
☞ CONNECTION FOR ENCODER INPUT SIGNAL	- 17 -
☞ CONNECTION FOR DIGITAL INPUT	- 17 -
☞ CONNECTION FOR DIGITAL OUTPUT	- 20 -
CHAPTER 4 SOFTWARE INSTALLATION.....	- 23 -
☞ DRIVE INSTALLATION IN WIN2000	- 23 -
☞ DRIVE INSTALLATION UNDER WINXP	- 26 -
CHAPTER 5 FUNCTIONS.....	- 28 -
☞ PULSE OUTPUT METHOD	- 29 -
☞ HARDWARE LIMIT SIGNAL	- 29 -
☞ LINEAR INTERPOLATION	- 29 -
☞ QUANTITATIVE DRIVING	- 30 -
☞ VELOCITY CURVE.....	- 31 -
☞ POSITION LOCK	- 33 -
☞ EXTERNAL SIGNAL DRIVING	- 33 -

CHAPTER 6 LIST OF ADT8940A1 BASIC LIBRARY FUNCTIONS... - 33 -**CHAPTER 7 DETAILS OF ADT8940A1 BASIC LIBRARY FUNCTIONS - 36 -**

- ☞ CATEGORY OF BASIC PARAMETER SETTING - 36 -
- ☞ CATEGORY OF DRIVE STATUS CHECK - 39 -
- ☞ CATEGORY OF MOVEMENT PARAMETER SETTING - 40 -
- ☞ CATEGORY OF MOTION PARAMETER CHECK - 42 -
- ☞ CATEGORY OF DRIVE - 43 -
- ☞ CATEGORY OF SWITCH AMOUNT INPUT/ OUTPUT - 45 -
- ☞ CATEGORY OF COMPOSITE DRIVING - 45 -
- ☞ CATEGORY OF EXTERNAL SIGNAL DRIVING - 49 -
- ☞ CATEGORY OF LOCK POSITION - 50 -
- ☞ CATEGORY OF HARDWARE CACHE - 51 -

CHAPTER 8 GUIDE TO MOTION CONTROL FUNCTION LIBRARY - 53 -**CHAPTER 9 BRIEFING ON MOTION CONTROL DEVELOPMENT- 56 -**

- ☞ CARD INITIALIZATION - 56 -
- ☞ SPEED SETTING - 56 -
- ☞ STOP0, STOP1 SIGNAL - 57 -

CHAPTER 10 PROGRAMMING SAMPLES IN MOTION CONTROL DEVELOPMENT - 58 -

- ☞ VB PROGRAMMING SAMPLES - 58 -
- ☞ VC PROGRAMMING SAMPLES - 81 -

CHAPTER 11 NORMAL FAILURES AND SOLUTIONS - 96 -

- MOTOR SERVICE FAILURE - 97 -



☞ ABNORMAL SWITCH AMOUNT INPUT - 99 -

APPENDIX A TYPICAL WIRING FOR MOTOR DRIVER..... - 101 -

APPENDIX B INTRODUCTION ON APPLICABLE LIBRARY 错误!

未定义书签。

Chapter 1 General information

INTRODUCTION

ADT8940A1 Card is a kind of high-performance 4-axis servo/ stepping control card based on PCI bus and supporting Plug & Play, while one system can support up to 16 control cards and control up to 64 lines of servo/ stepping motors.

Pulse output method may be single pulse (pulse + direction) or double pulse (pulse+pulse), with the maximum pulse frequency of 2MHz. Advanced technologies are applied to ensure the frequency tolerance is less than 0.1% despite of high output frequency.

It supports 2-4 axis of linear interpolation, with the maximum interpolation speed of 1MHz.

External signal (handwheel or general input signal) driving can be either constant or continuous driving

With position lock, you can lock the value of logical counter or actual position counter.

Speed can be set as constant speed or trapezoidal acceleration/ deceleration.

Hardware caching features with a large-capacity.

I / O response time of about 500μs.

Position management is realized through two up/ down counters, one used to manage logical positions of internally driven pulse output, and the other used to receive external input, with encoder or grating ruler inputted through A/ B phase as the input signal.

Counters are up to 32 digits, specially, the range is 2,147,483,648~+2,147,483,647.

The system also provides DOS/WINDOWS95/98/NT/2000/XP/WINCE development libraries and enable software development in VC++, VB, BC++, LabVIEW, Delphi, and C++Builder.

MAIN FEATURES

- 32-digit PCI bus, enabling Plug & Play
- All the input and output are under photoelectric coupler isolation, with strong resistance to disturbance.
- 4-axis servo/ stepping motor control, with every axis able to move independently without mutual effects.

- Frequency tolerance for pulse output is less than 0.1%.
- The maximum pulse output frequency is 2MHz.
- Pulse output may be single (pulse+ direction) or double(pulse+ pulse)
- All the 4 axes have position feedback input in 32-digit counting, giving the maximum counting range of -2,147,483,648~ +2,147,483,647.
- Trapezoidal acceleration/ deceleration
- 2-4 axis linear interpolation.
- Maximum interpolation speed: 1MHz.
- I/O response time of about 500μs.
- Handwheel and external signal operation, position lock, large-capacity of hardware caching.
- Real-time reading of logical, real and driving speeds during movement
- 40-line digital input (each axis of position feedback may be used as 2 input points, altogether 8).
- Two limit input for each axis may be set as Nil and work as general input
- Up to 16 control cards supported within one system.
- DOS/WINDOWS95/98/NT/2000/XP/WIN CE supported.

**APPLICATIONS**

- ✎ Multi-axis engraving system
- ✎ Robot system
- ✎ Coordinate measurement system
- ✎ PC-based CNC system

Chapter 2 Hardware installation

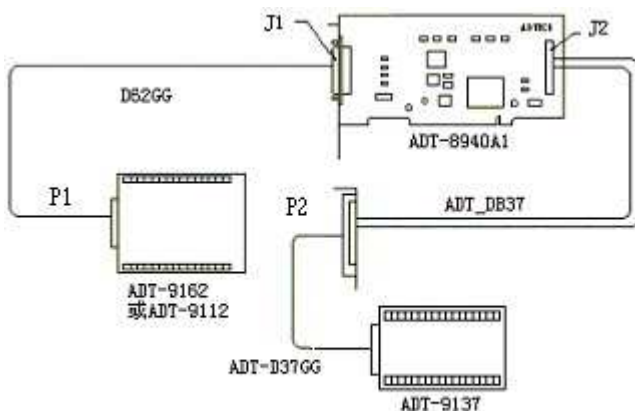
PARTS

1. ADT-8940A1 User Manual (this manual)
2. ADT-8940A1 4-axis PCI bus high-performance motion control card
3. ADT-8940A1 user CD
4. ADT-DB37 1 pc
5. ADT-D37GG 1 pc
6. ADT-9137 37-pin signal connecting plate, 1 pcs
7. D62GG
8. ADT-9162 connecting plate, 1 pcs
9. ADT-9112 connecting plate, 1 pcs(Optional configuration, 62-pin terminal block)

INSTALLATION

1. Switch off the computer power supply (for ATX supply case, switch off the overall power)
2. Open the back cover of the computer case
3. Insert ADT-8940 into an available PCI slot
4. Ensure the golden finger of ADT-8940 has been fully inserted the slot and then fasten card with screws
5. Connect one end of the D62GG cable to J1 interface of motion card and the other end to terminal block ADT_9162.
6. Check whether it is necessary to install J2 interface cable. To install J2 if necessary:(1) Connect one end of ADT-DB37 to J2 of motion card and the other end to P2 of ADT-DB37;(2) Fix the ADT-DB37 on the rear side of the enclosure;(3) Connect ADT-D37GG to P2 of the transition board and ADT-D37GG.

Chapter 3 Electrical connection



There are two input/ output interfaces inside an ADT8940A1 card, whereby J1 is for 62-pin socket and J2 is for 25-pin.

J1 is the signal cable for pulse output of X, Y, Z and A axis, switch amount input and switch amount output (OUT0-OUT11); **J2** is the signal cable for encoder input and switch amount input of X, Y, Z and A axis; switch amount input and switch amount output (OUT12-OUT15).

Signals are defined as follows:

👉 J1 line

PUCOM1	1	32	IN11(XEXP-)
XPU+/CW+	2	33	IN12(ZLMT-)
XPU-/CW-	3	34	IN13(ZLMT+)
XDR+/CCW+	4	35	IN14(ZSTOP0)
XDR-/CCW-	5	36	IN15(ZSTOP1)
YPU+/CW+	6	37	IN16(ZEXP+)
YPU-/CW-	7	38	IN17(ZEXP-)
YDR+/CCW+	8	39	IN18(ADM-)
YDR-/CCW-	9	40	IN19(ADM+)
PUCOM2	10	41	IN20(AS0P0)
ZPU+/CW+	11	42	IN21(AS0P1)
ZPU-/CW-	12	43	IN22(AEXP+)
ZDR+/CCW+	13	44	IN23(AEXP-)
ZDR-/CCW-	14	45	OUT0
APU+/CW+	15	46	OUT1
APU-/CW-	16	47	OUT2
ADR+/CCW+	17	48	OUT3
ADR-/CCW-	18	49	OUTCOM1
INCOM1	19	50	OUT4
IN0(XLMT-)	20	51	OUT5
IN1(XLMT+)	21	52	OUT6
IN2(XSTOP0)	22	53	OUT7
IN3(XSTOP1)	23	54	OUTCOM2
IN4(XEXP+)	24	55	OUT8
IN5(XEXP-)	25	56	OUT9
IN6(YLMT-)	26	57	OUT10
IN7(YLMT+)	27	58	OUT11
INCOM2	28	59	OUTCOM3
IN8(YSTOP0)	29	60	+12V
IN9(YSTOP1)	30	61	GND
IN10(YEXP+)	31	62	

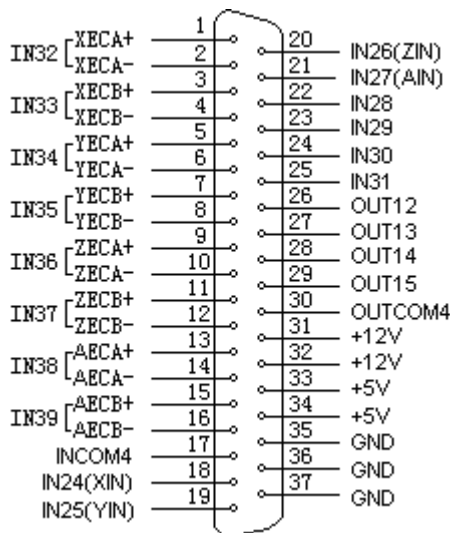
Line number	Signal	Introduction
1	PCOM1	Used for single-port input, not available for external power supply
2	XPU+/CW+	X pulse signal +
3	XPU-/CW-	X pulse signal -
4	XDR+/CCW+	X direction signal +
5	XDR-/CCW-	X direction signal -
6	YPU+/CW+	Y pulse signal +
7	YPU-/CW-	Y pulse signal -
8	YDR+/CCW+	Y direction signal +
9	YDR-/CCW-	Y direction signal -
10	PCOM2	Used for single-port input, not available for external power supply

11	ZPU+/CW+	Z pulse signal +
12	ZPU-/CW-	Z pulse signal -
13	ZDR+/CCW+	Z direction signal +
14	ZDR-/CCW-	Z direction signal -
15	APU+/CW+	A pulse signal +
16	APU-/CW-	A pulse signal -
17	ADR+/CCW+	A direction signal +
18	ADR-/CCW-	A direction signal -
19	INCOM1	Common for pin20-27 pin (Input points for switch amount)
20	IN0(XLMT-)	Limit- signal for X,able to work as general input signal
21	IN1(XLMT+)	Limit+ signal for X, able to work as general input signal
22	IN2 (XSTOP0)	STOP0- signal for X,able to work as general input signal
23	IN3 (XSTOP1)	STOP1- signal for X, able to work as general input signal
24	IN4(XEXP+)	Positive direction of the Manually Signal for X,able to work as general input signal
25	IN5(XEXP-)	negative direction of the Manually Signal for X, able to work as general input signal
26	IN6 (YLMT-)	Limit- signal for Y, able to work as general input signal
27	IN7 (YLMT+)	Limi+ signal for Y, able to work as general input signal
28	INCOM2	Common for pin29-36 (Input points for switch amount)
29	IN8 (YSTOP0)	STOP0- signal for Y,able to work as general input signal

30	IN9 (YSTOP1)	STOP1- signal for Y, able to work as general input signal
31	IN10(YEXP+)	Positive direction of the Manually Signal for Y,able to work as general input signal
32	IN11(YEXP-)	negative direction of the Manually Signal for Y able to work as general input signal
33	IN12(ZLMT-)	Limit_signal for Z,able to work as general input signal
34	IN13(ZLMT+)	Limit+ signal for Z, able to work as general input signal
35	IN14(ZSTOP0)	STOP0- signal for Z,able to work as general input signal
36	IN15(ZSTOP1)	STOP1- signal for Z, able to work as general input signal
37	INCOM3	Common for pin38-45 (Input points for switch amount)
38	IN16(ZEXP+)	Positive direction of the Manually Signal for Z,able to work as general input signal
39	IN17(ZEXP-)	negative direction of the Manually Signal for Z,able to work as general input signal
40	IN18(ALMT-)	Limit- signal for A,able to work as general input signal
41	IN19(ALMT+)	Limit+ signal for A, able to work as general input signal
42	IN20(ASTOP0)	STOP0- signal for A,able to work as general input signal
43	IN21(ASTOP1)	STOP1- signal for A, able to work as general input signal
44	IN22(AEXP+)	Positive direction of the Manually Signal for A,able to work as general input signal

45	IN23(AEXP-)	negative direction of the Manually Signal for A, able to work as general input signal
46	OUT0	Output points for switch amount
47	OUT1	Output points for switch amount
48	OUT2	Output points for switch amount
49	OUT3	Output points for switch amount
50	OUTCOM1	General negative common for Output0-3 (Output points for switch amount)
51	OUT4	Output points for switch amount
52	OUT5	Output points for switch amount
53	OUT6	Output points for switch amount
54	OUT7	Output points for switch amount
55	OUTCOM2	General negative common for Output4-7 (Output points for switch amount)
56	OUT8	Output points for switch amount
57	OUT9	Output points for switch amount
58	OUT10	Output points for switch amount
59	OUT11	Output points for switch amount
60	OUTCOM3	General negative common for Output8-11 (Output points for switch amount)
61	+12V	Positive port of internal +12V power supply, not available for external power supply
62	GND	Internal power supply earthing

J2 line



Line number	Signal	Introduction
1	XECA+	X-axis encoder A-phase input+
2	XECA-	X-axis encoder A-phase input -, able to work as general input signal 32
3	XECB+	X-axis encoder B-phase input +
4	XECB-	X-axis encoder B-phase input -, able to work as general input signal 33
5	YECA+	Y-axis encoder A-phase input+
6	YECA-	Y-axis encoder A-phase input -, able to work as general input signal 34
7	YECB+	Y-axis encoder B-phase input +
8	YECB-	Y-axis encoder B-phase input -, able to work as general input signal 35
9	ZECA+	Z-axis encoder A-phase input+
10	ZECA-	Z-axis encoder A-phase input -, able to work as general input signal 36
11	ZECB+	Z-axis encoder B-phase input +
12	ZECB-	Z-axis encoder B-phase input -, able to work as general input signal 37
13	AECA+	A-axis encoder A-phase input+

14	AECA-	A-axis encoder A-phase input -, able to work as general input signal 38
15	AECB+	A-axis encoder B-phase input +
16	AECB-	A-axis encoder B-phase input -, able to work as general input signal 39
17	INCOM4	Common for pin18-25 (Input points for switch amount)
18	IN24(XIN)	X position lock signal; can be used as universal input signal
19	IN25(YIN)	Y position lock signal; can be used as universal input signal
20	IN26(ZIN)	Z position lock signal; can be used as universal input signal
21	IN27(AIN)	A position lock signal; can be used as universal input signal
22	IN28	General input signal
23	IN29	General input signal
24	IN30	General input signal
25	IN31	The signal to stop using the hardware, able to work as general input signal
26	OUT12	Output points for switch amount
27	OUT13	Output points for switch amount
28	OUT14	Output points for switch amount
29	OUT15	Output points for switch amount
30	OUTCOM4	General negative common for Output12-15 (Output points for switch amount)
31	+12V	Positive port of internal +12V power supply, not available for external power supply
32	+12V	Positive port of internal +12V power supply, not available for external power supply
33	+5V	Positive port of internal +5V power supply, not available for external power supply
34	+5V	Positive port of internal +5V power supply, not available for external power supply
35	GND	Internal power supply earthing
36	GND	Internal power supply earthing
37	GND	Internal power supply earthing

Remark: In case an encoder is used for general input signals, XECA+, XECB+, YECA+, YECB+, ZECA+, ZECB+, AECA+, and AECB+ will be respectively used as

public ports of corresponding input signals.

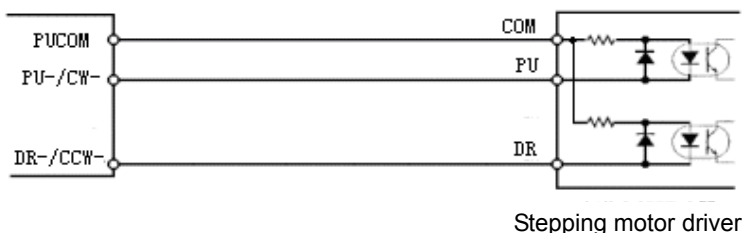
Voltage at the public ports can only be +5V; in case of using an external+12V power supply, users must serially connect a 1K resistance. Please refer to the following digital input connection part for wiring method.

✎ CONNECTION FOR PULSE/ DIRECTION INPUT SIGNAL

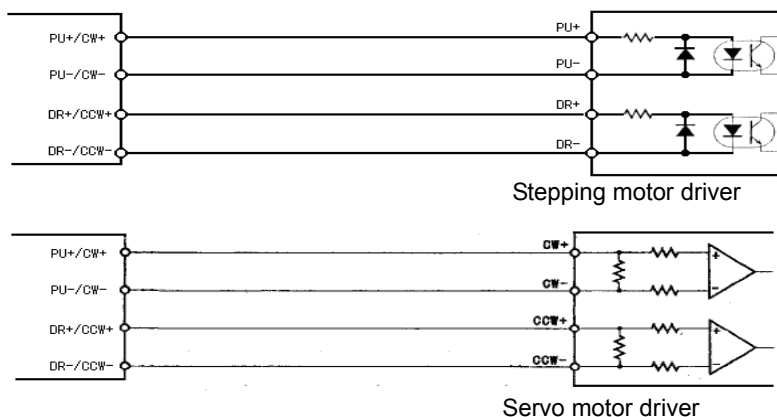
Pulse output is in differential output.

May be conveniently connected with a stepping/ servo driver

The following figure shows open-collector connection between pulse and direction.



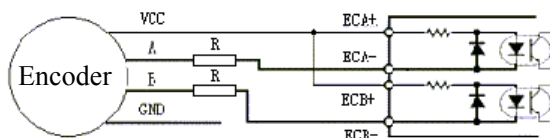
The following figure shows differential-output connection between pulse and direction signals; this method is recommended as it is differential connection with strong resistance to disturbance.



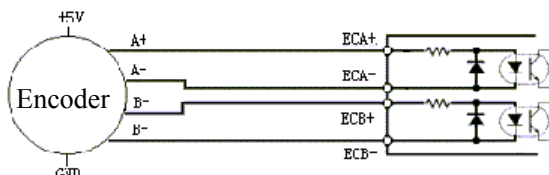
Remark: Refer to Appendix A for wiring maps of stepping motor drivers, normal servo

motor driver and terminal panel.

✎ CONNECTION FOR ENCODER INPUT SIGNAL

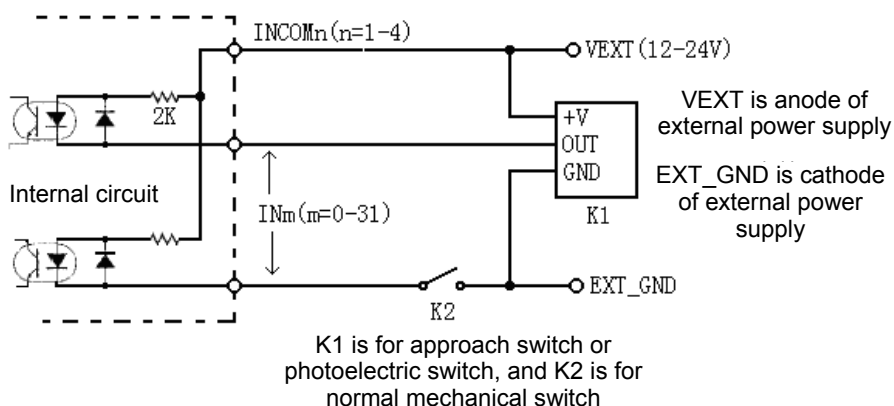


Wiring map for an open-collect output-type encoder. For +5V power supply, R is not required; for +12V power supply, R= 1K Ω ; and for +24V power supply, R= 2K Ω



Wiring map for a differential-driver output-type encoder

✎ CONNECTION FOR DIGITAL INPUT



Remark:

(1) Public terminal for IN0-IN7: INCOM1

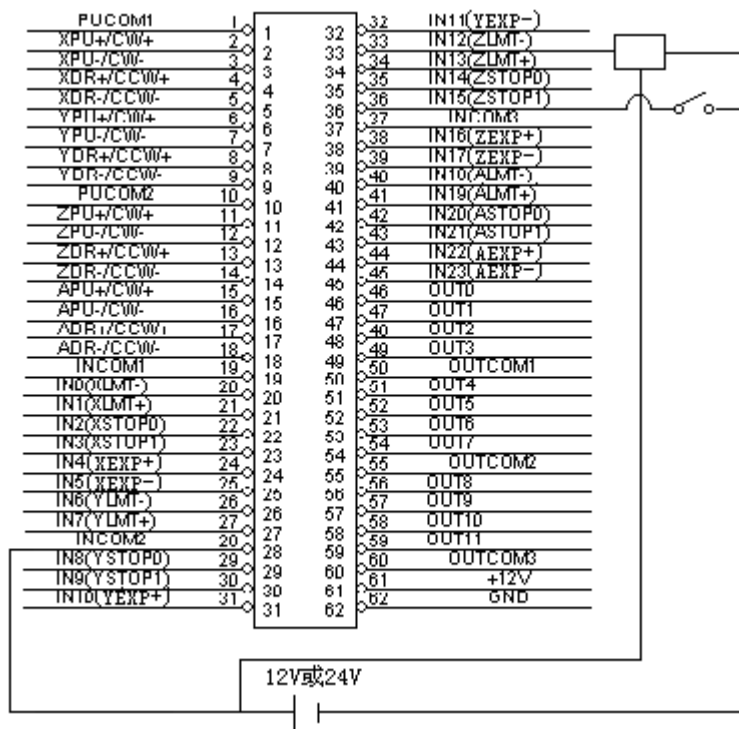
Public terminal for IN8-IN15: INCOM2

Public terminal for IN16-IN25: INCOM3

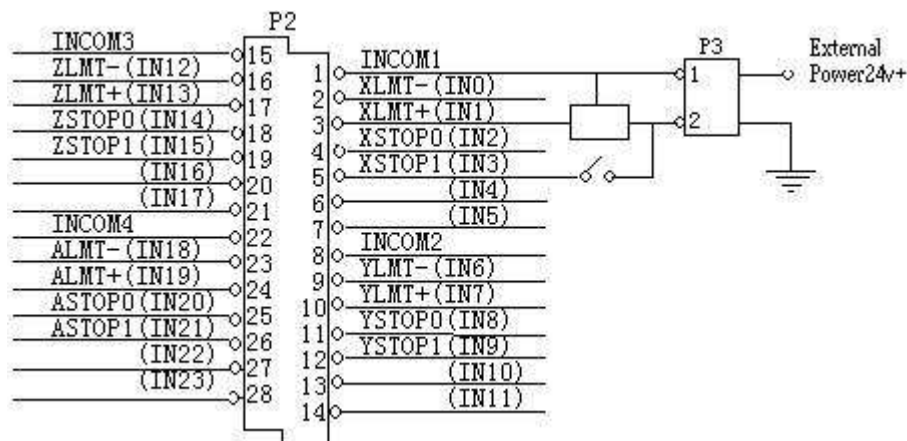
Public terminal for IN24-IN31: INCOM4

- (2) To make input signals effective, users shall make sure: firstly, the photoelectric coupling public ports for corresponding input signals (INCOM1, INCOM2, INCOM3 or INCOM4) have been connected with anodes of 12V/ 24V power supply; secondly, one port of the normal switch or earthing cable of the approach switch has been connected with the cathode (earthing cable); and lastly, the other port of the normal switch or the control of the approach switch has been connected with the input port corresponding by the terminal panel.
- (3) The following is the actual wiring map of power supply from normal switch and approach switch to photoelectric coupling public ports, through external power supply.

☞ **ADT-9162 terminal block wiring diagram:**



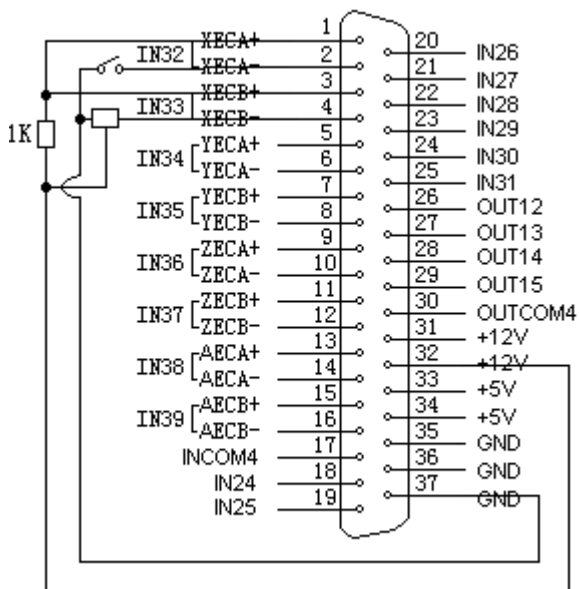
ADT-9112 terminal block wiring diagram:



Remark:

When the jumper cap T1, T2 used in parallel, the four INCOM ports were connected to the unified 24V power when the jumper cap connected. Do not need to connect the pin1、8、15、22 which are on P2 Wiring terminals to the +24 power.

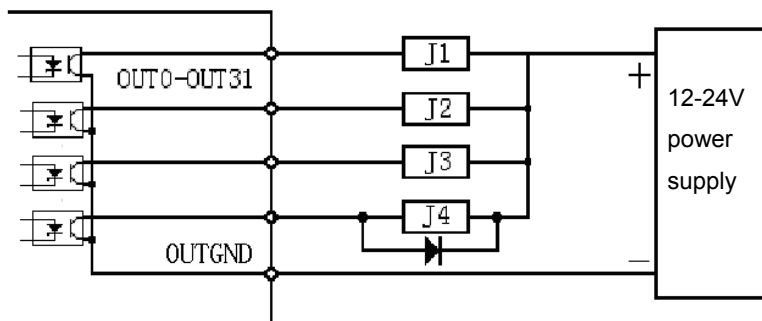
☞ **Encoder signals used as the general-purpose input signals wiring diagram**



Remark: In case an encoder is used for general input signals, XECA+, XECB+, YECA+, YECB+, ZECA+, ZECB+, AECA+, and AECB+ will be respectively used as public ports of corresponding input signals.

Voltage at the public ports can only be +5V; in case of using an external +12V power supply, users must serially connect a 1K resistance. Please refer to the following digital input connection part for wiring method.

☞ **CONNECTION FOR DIGITAL OUTPUT**



For inductive loading such as relay, add continuous diode at the two ends of the loading, as shown in J4

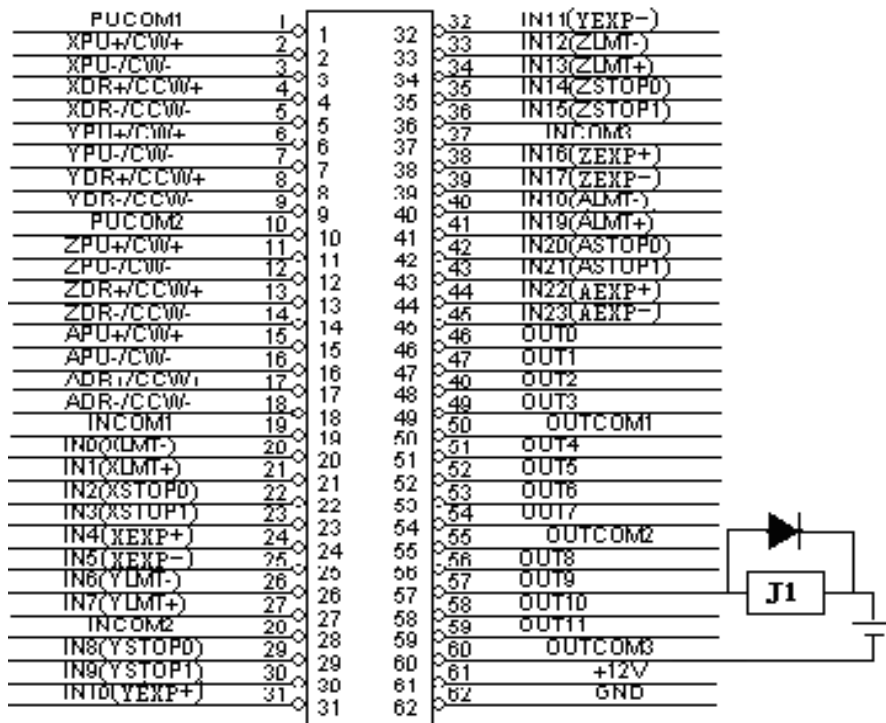
Remark:

(1) Public terminal for OUT0-OUT5: OUTCOM1

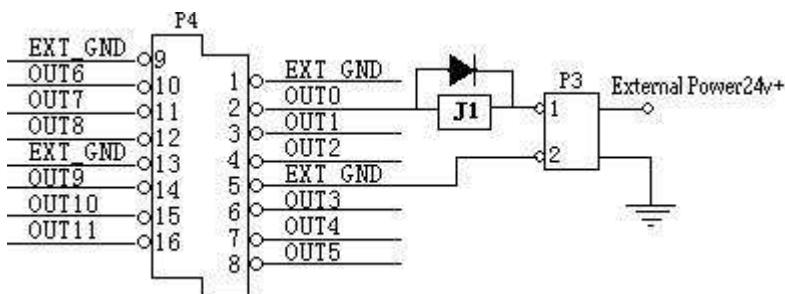
(2) To make output signals effective, users must make sure of connection between the output public port OUTCOM1 and cathode of external power supply (earthling cable) if using external power supply, or connection between internal power supply earthling (GND) and the ground if using internal power supply. Relay coils must have one side connected with the power supply anode and the other side connected with the corresponding output port of the terminal panel.

(3) The following picture is the actual wiring map for power supply by external power supply.

☞ **ADT-9162 terminal block wiring diagram:**



ADT-9112 terminal block wiring diagram:



Remark:

General negative common for pin1, 5,9,13 on P4 has been grounded. Additionally, users can also connect them to ground. Please connect the load between the power supply and output contact. The load can be resistive, inductive or capacitive. The digital

output part of ADT-9112 terminal used power amplifier circuit design; output currents up to 500MA.It can be direct-drive cylinders, solenoid valves and other devices, do not need to connect it to the external +24 power.

Chapter 4 Software installation

ADT8940A1 card must be used with drive installed under Win95/ Win98/ NT/ Win2000/ WinXP, but in case of DOS, no drive is required to be installed.

The following part takes Win2000 and WinXP for example, and users may refer to other operating systems.

Drive for the control card is located in the Drive/ ControlCardDrive folder within the CD, and the drive file is named as ADT8940A1.INF.

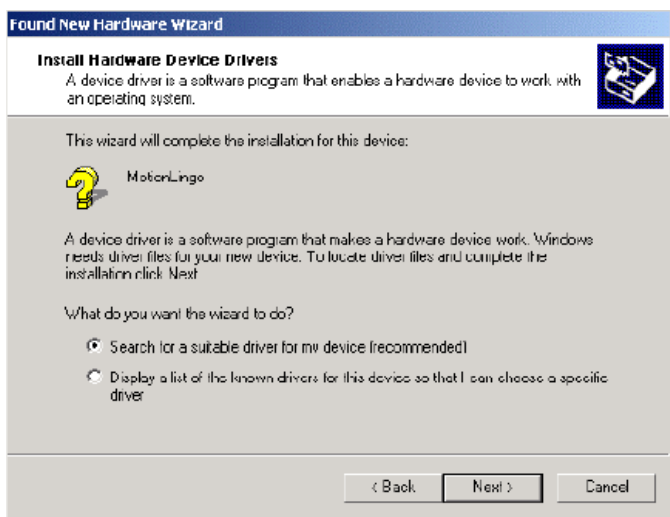
☛ DRIVE INSTALLATION IN WIN2000

The following part takes Win2000 Professional Version as example to indicate installation of the drive; other versions of Win2000 are similar.

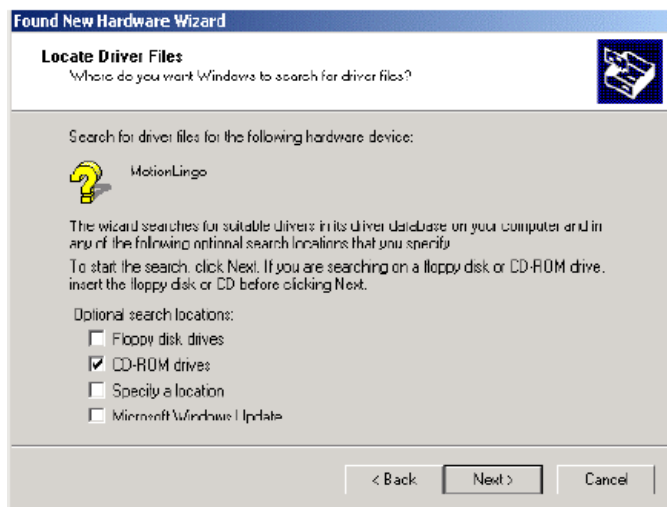
After attaching the ADT8940A1 card to the PCI slot of a computer, a user shall log in as administrator to the computer; upon display of the initial interface, the computer shall notify "Found new hardware" as follows:



Just click “Next” to display the following picture:



Click again “Next” to display the following picture:



Then select “Specify a location” and Click again “Next” and Click “Browse” button to select DevelopmentPackage/ Drive/ CardDrive and find the ADT8940A1.INF file, then click “OK” to display the following interface:



Click “Next” to display the following picture:

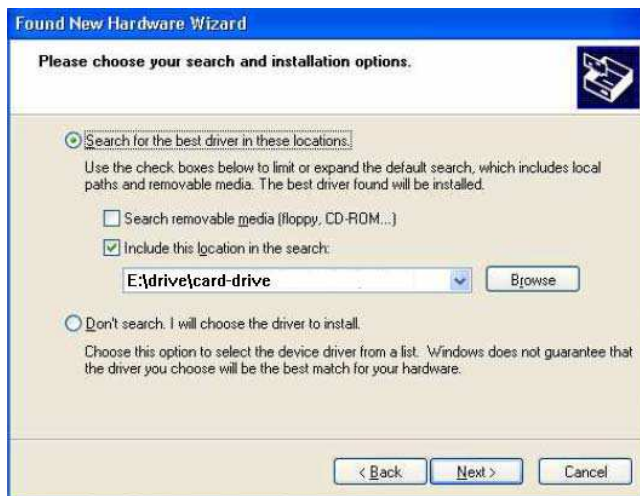


Finally click “Finish” to complete installation.

DRIVE INSTALLATION UNDER WINXP

Installation under WinXP is similar to that under Win2000, specifically:





Click “Browse” button to select Drive/ CardDrive and find the ADT8940A1.INF file, then click “Next” to display the following interface:



Then click “Finish” to complete installation.

Chapter 5 Functions

Pulse output method

Pulse output may be realized through either independent 2-pulse or 1-pulse. In case of independent 2-pulse, the positive direction drive has PU/CW outputting drive pulses, and the negative direction drive has DR/CCW outputting drive pulses. In case of 1-pulse, PU/CW outputs drive pulse and DR/CCW outputs direction signals.

Pulse/ direction is set on the positive logical level

Pulse output type	Drive direction	Output signal waveform	
		PU/CW signal	DR/CCW signal
Independent 2-pulse	+Direction		Low level
	-Direction	Low level	
1-pulse 1-direction	+Direction		Low level
	-Direction		Hi level

Hardware Limit signal

Hardware limit signals LMT+ and LMT- are respectively to limit the input signals outputted by drive pulse along positive and negative directions, able to be set as “effective”, “ineffective” with high/ low levels.—Actually “effective” or “ineffectively” can be set for positive limit and negative limit individually; in case “ineffective” is selected, they may work as ordinary input points.

Hardware limit signals STOP0 and STOP1 are input signals that may realize hardware termination for all axis drive and may be set as “effective”, “ineffective” as well as the termination method for high/ low levels. In case “ineffective” is selected, they may work as general input points. Besides, they, when working as drive for interpolation, are effective for the minimum interpolation axis only.

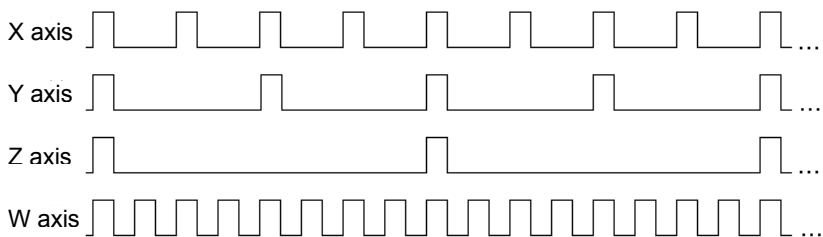
Linear interpolation

This card may work for 2-4 axes linear interpolation and support any 2 axes or 3 axes linear interpolation, under the modified method of point-by-point comparison, which can ensure uniform pulse along the long axis, giving the precision within one pulse.

Firstly, take the axis outputting the maximum pulses among the axes joining interpolation as the long axis, and proportionally distribute for the rest axes. Speed control applies only to speed of the long axis, for example: (1-X axis, 2-Y axis, 3-Z axis, and

4-W axis)

Take four axes for linear interpolation, while Axis 1 outputs 1000 pulses, Axis 2 outputs 500, Axis 3 outputs 250 and Axis outputs 2000.



From the above figure, W axis (Axis 4) shall be the long axis, and the rest axes proportionally share the pulses.

Setting of interpolation speed takes the minimum speed of an axis among the joined axes as the benchmark, for example, if Axis 2 and Axis 3 join linear interpolation, the interpolation speed will be determined by speed of Axis 2. Moreover, speed of interpolation is only half of the single axis. Example in more details:

Axis 2 and Axis 3 work for two-axis linear interpolation, with Axis 2 outputs 10000 pulses along positive direction and Axis 3 outputs 5000 pulses along negative direction, which means Axis 2 is the long axis.

```
set_startv(0,2,1000);
set_speed(0,2,1000);
inp_move2(0,2,3,10000,-5000);
```

After execution of the above program, Axis 2 will send 10000 pulses in the frequency of $1000/2=500\text{Hz}$, while frequency of Axis 3 shall be $500*5000/10000=250\text{Hz}$.

If speed of Axis 2 realizes trapezoidal acceleration/ deceleration, interpolation will also follow such trapezoidal acceleration/ deceleration.

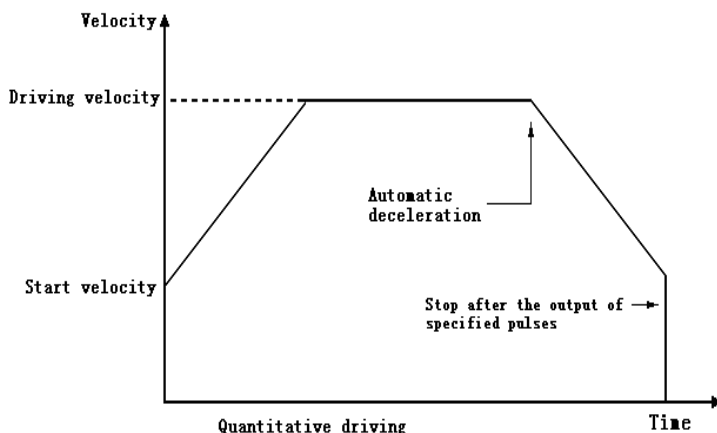
Quantitative driving

Quantitative driving means to output pulse of specified amount in constant velocity or acceleration/deceleration. It is useful to move to specified position or execute specified action. The quantitative driving of acceleration/deceleration is shown in the following picture. Deceleration starts when left output pulses are less than accumulated acceleration pulses. The driving stops after the output of specified

pulses.

Configure the following parameters to execute the quantitative driving of acceleration/deceleration:

- a) Acceleration/deceleration A/D
- b) Start velocity SV
- c) Driving velocity V
- d) Output pulse P



Acceleration/deceleration quantitative driving automatically decelerates from the deceleration point as shown in the picture above.

Velocity curve

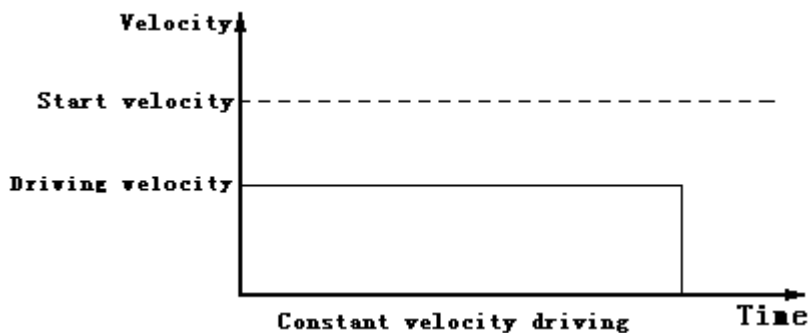
1) Constant velocity driving

Linear acceleration/deceleration driving is to accelerate from start velocity to specified driving velocity linearly.

In quantitative driving, the acceleration counter records the accumulated pulses of acceleration. If left output pulses are less than acceleration pulses, it will decelerate (automatically). In deceleration, it will decelerate to start velocity linearly in specified velocity.

Configure the following parameters to execute linear acceleration/deceleration driving:

- Range R
- Acceleration A Acceleration and deceleration
- Deceleration D Deceleration if they are set separately (if necessary)
- Start velocity SV
- Driving velocity V



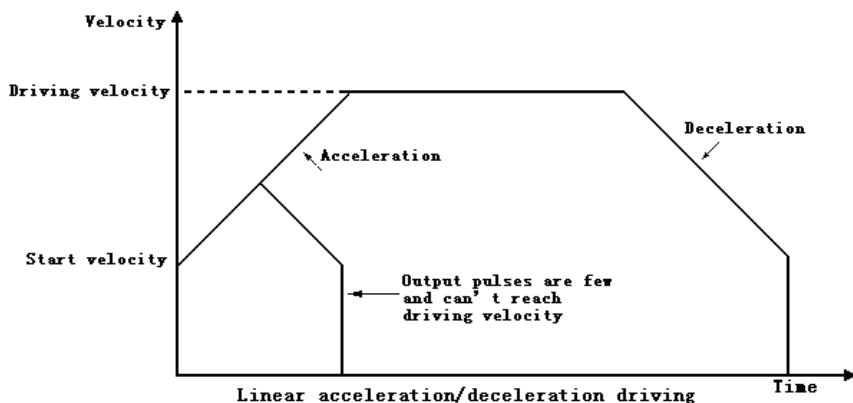
2) Linear acceleration/deceleration driving

Linear acceleration/deceleration driving is to accelerate from start velocity to specified driving velocity linearly.

In quantitative driving, the acceleration counter records the accumulated pulses of acceleration. If left output pulses are less than acceleration pulses, it will decelerate (automatically). In deceleration, it will decelerate to start velocity linearly in specified velocity.

Configure the following parameters to execute linear acceleration/deceleration driving:

- Range R
- Acceleration A Acceleration and deceleration
- Deceleration D Deceleration if they are set separately (if necessary)
- Start velocity SV
- Driving velocity V



Position lock

Realize hardware position lock function with the IN signal on each axis. With one lock signal, the current position (either logical or actual) of all axes can be locked.

The position lock is useful in measuring system.

External signal driving

External signal driving is the motion controlled by external signals (handwheel or switch). It is mainly used in the manual debugging of machines and provides a lot of convenience in teaching system.

To simplify the wiring, the motion card short connects the positive driving signals of the four axes and also short connects the negative driving signals of the four axes; therefore, only one signal cable of the coder is connected to the external interface of external signals.

Chapter 6 List of ADT8940A1 basic library functions

List of V110 library functions

Function type	Function name	Function description	Page
Basic parameters	ADT8940A1_initial	Initialize card	37
	get_lib_version	Get version	37
	set_pulse_mode	Set pulse mode	37
	set_limit_mode	Set limit mode	38
	set_stop0_mode	Set stop mode	38
	set_stop1_mode	Set stop mode	39
	set_delay_time	Delay status	39
	set_suddenstop_mode	Hardware stop	39
Check for drive status	get_status	Get status of single-axis drive	40

	get_inp_status	Get status of interpolation	40
	get_delay_status	Delay status	40
	get_hardware_ver	Hardware version	40
Movement parameter setting	set_acc	Set acceleration	41
	set_startv	Set starting speed	41
	set_speed	Set drive speed	41
	set_command_pos	Set logical position counter	41
	set_actual_pos	Set real position counter	42
	set_symmetry_speed	Set symmetry speed	42
Check for motion parameters	get_command_pos	Get logical position	42
	get_actual_pos	Get real position	43
	get_speed	Get drive speed	43
	get_out	Get output status	43
Drive category	pmove	Single-axis quantitative drive	43
	dec_stop	Deceleration stop	44
	sudden_stop	Sudden stop	44
	lnp_move2	2-axis linear interpolation	45
	lnp_move3	3-axis linear interpolation	45

	inp_move4	4-axis linear interpolation	45
Switch amount category	read_bit	Read single input point	46
	write_bit	Output single output point	46
Composite driving	symmetry_relative_move	Symmetrical relative movement of single-axis	46
	symmetry_absolute_move	Symmetrical absolute movement of single-axis	47
	symmetry_relative_line2	Relative movement of two-axis symmetrical linear interpolation	47
	symmetry_absolute_line2	Two axes symmetric linear interpolation absolute moving	47
	symmetry_relative_line3	Three axes symmetric linear interpolation relative moving	48
	symmetry_absolute_line3	Three axes symmetric linear interpolation absolute moving	48
	symmetry_relative_line4	Four axes symmetric linear interpolation relative moving	49
	symmetry_absolute_line4	Four axes symmetric linear interpolation absolute moving	49
manual driving	manual_pmove	Quantitative drive function of external signal	50
	manual_continue	Continuous drive function of external signal	50
	manual_disable	Shut down the enabling of external signal drive	50
position lock	set_lock_position	set lock mode	50

	get_lock_status	get lock status	51
	get_lock_position	get lock position	51
	clr_lock_status	clean lock position	51
hardware cache	fifo_inp_move1	single axis FIFO	51
	fifo_inp_move2	two axes FIFO	52
	fifo_inp_move3	three axes FIFO	52
	fifo_inp_move4	four axes FIFO	53
	reset_fifo	reset FIFO	53
	read_fifo_count	read FIFO	53
	read_fifo_empty	read FIFO	53
	read_fifo_full	read FIFO	54

Chapter 7 Details of ADT8940A1 basic library functions

☛ CATEGORY OF BASIC PARAMETER SETTING

1.1 Initialize card

```
int ADT8940A1_initial(void);
```

- (1) Return >0 means amount of installed ADT8940A1 cards; in case the Return is 3, the available card numbers shall be 0, 1, and 2;

- (2) Return =0 means no installation of ADT8940A1 card;
 (3) Return <0 means no installation of service if the value is -1 or PCI bus failure is the value is -2.

Remark: Initialization functions are preliminary conditions to call other functions, thus must be called firstly so as to verify available cards and initialize some parameters.

1.2 Get current library version

```
int get_lib_version();
```

Here return are combination of hardware and library version number.

1.3 Set output pulse mode

```
int set_pulse_mode(int cardno, int axis, int value,int logic,int dir_logic);
```

cardno Card number

axis Axis number (1-4)

value 0: Pulse + Pulse method 1: Pulse + direction method

Pulse/ direction are both of positive logic setting

Pulse output method	Drive direction	Output signal waveform	
		PU/CW signal	DR/CCW signal
Independent 2-pulse method	Positive drive output		Low level
	Negative drive output	Low level	
1-pulse method	Positive drive output		Low level
	Negative drive output		Hi level

Positive logic pulse : 

Negative logic pulse : 

dir-logic 0: Positive logic direction input signal 1: Negative logic direction input signal

dir_logic	Positive direction logic pulse	Negative direction logic pulse
0	Low	Hi
1	Hi	Low

Default mode: Pulse + direction, with positive logic pulse and positive logic direction input signal

Default mode: positive and negative limits with low level are effective

Default mode: stop0 is ineffective

Default mode: stop1 is ineffective

1.7 Set mode of stop1 input signal

int set_delay_time(int cardno, long time)

cardno Card number

time delay time (Unit: 1/8us)

Return 0: Correct 1: Wrong

Remark: The time unit is 1/8us, with the maximum *integer* value as its maximum

1.8 Set stop using the hardware

int set_suddenstop_mode(int cardno, int v, int logical)

cardno Card number

v 0: ineffective 1: effective

logical 0: low level effective 1: high level effective

Return 0: Correct 1: Wrong

Remark: Hardware stop signals are assigned to use the 34 pin at the P3 terminal panel (IN31)

CATEGORY OF DRIVE STATUS CHECK

2.1 Get status of single-axis drive

int get_status(int cardno, int axis, int *value)

cardno Card number

axis Axis number (1-4)

value Indicator of drive status

0: Drive completed

Non-0: Drive in process

Return 0: Correct 1: Wrong

2.2 Get status of Interpolation

int get_inp_status(int cardno, int *value)

cardno Card number

value Indicator of i Interpolation:

0: Interpolation completed 1: Interpolation in process
Return 0: Correct 1: Wrong

2.3 Get status of Delay

int get_delay_status(int cardno)

cardno Card number

Return 0: delay completed 1: delay in process

2.4 Get hardware version

int get_hardware_ver(int cardno)

cardno Card number

Return 256: version 1.0 257: version 1.1

CATEGORY OF MOVEMENT PARAMETER SETTING

🔴* **Remark:** The following parameters are not determined after initialization thus must be set before use.

3.1 Set acceleration

int set_acc(int cardno,int axis,long value);

cardno Card number

axis Axis number

value Acceleration(0-32000)

Return 0: Correct 1: Wrong

3.2 Set starting speed

int set_startv(int cardno,int axis,long value);

cardno Card number

axis Axis number

value Speed(0-2M)

Return 0: Correct 1: Wrong

3.3 Set drive speed

int set_speed(int cardno,int axis,long value);

cardno Card number
axis Axis number
value Speed(0-2M)
Return 0: Correct 1: Wrong

3.4 Set logical position counter

This is to set values for the logical position counter

int set_command_pos(int cardno,int axis,long value);

cardno Card number
axis Axis number
value Range (-2147483648~+2147483647)
Return 0: Correct 1: Wrong

A logical position counter can read and write at any time.

3.5 Set real position counter

This is to set values for the real position counter

int set_actual_pos(int cardno,int axis,long value);

cardno Card number
axis Axis number
value Range (-2147483648~+2147483647)
Return 0: Correct 1: Wrong

An real position counter can read and write at any time.

3.6 Set symmetry speed

This is to set values for the symmetry speed

int set_symmetry_speed(int cardno, int axis,long lspd,long hspd,double tacc);

cardno Card number
axis Axis number
lspd start speed
hspd running speed
tacc acceleration time
Return 0: Correct 1: Wrong

Remark: This function is a combination of multiple functions,
such as set_acc,set_startv,set_speed etc.

CATEGORY OF MOTION PARAMETER CHECK

The following functions can be called at any time

4.1 Get logic position of each axis

int get_command_pos(int cardno,int axis,long *pos)

cardno Card number
axis Axis number
pos Indicator of logic position value
Return 0: Correct 1: Wrong

This function can get the logic position of the corresponding axis at any time, and in case of no out-step by motor, pos values just indicate the current position of the axis.

4.2 Get real position of each axis (i.e., encoder feedback input)

int get_actual_pos(int cardno,int axis,long *pos)

cardno Card number
axis Axis number
pos Indicator of actual position value
Return 0: Correct 1: Wrong

This function can get the real position of the corresponding axis at any time, and even though in case of out-step by motor; pos values still indicate the real position of the axis.

4.3 Get motion speed

int get_speed(int cardno,int axis,long *speed)

cardno Card number
axis Axis number
speed Indicator of current drive speed

Return 0: Correct 1: Wrong

Its data unit is same as that for motion speed setting value V.

This function can get the axis drive speed at any time.

4.4 Get status of output

int _stdcall get_out(int cardno, int number)

cardno Card number

axis Axis number

Return Current output point status -1: Wrong

CATEGORY OF DRIVE

5.1 Single-axis quantitative drive

int pmove(int cardno,int axis,long pulse)

cardno Card number

axis Axis number

pulse Outputted pulses

>0: move along positive direction

<0: move along negative direction

Range (-268435455~+268435455)

Return 0: Correct 1: Wrong

Remark: Users must correctly set the parameters required by speed curve before making drive commands.

5.2 Deceleration stop

int dec_stop(int cardno,int axis)

cardno Card number

axis Axis number

Return 0: Correct 1: Wrong

During drive pulse output, this command will make deceleration stop. Users may also use this command to stop when the drive speed is lower than the

starting speed.

Remark: During linear interpolation, if requiring deceleration stop, users shall make command only for the earliest interpolation axis, otherwise may fail to achieve expected results.

5.3 Sudden stop

int sudden_stop(int cardno,int axis)

cardno Card number

axis Axis number

Return 0: Correct 1: Wrong

This command will suddenly stop the pulse output in process, even though it is in acceleration/ deceleration drive.

Remark: During linear interpolation, if requiring sudden stop, users shall make command only for the earliest interpolation axis, otherwise may fail to achieve expected results.

5.4 2-axis interpolation

int inp_move2(int cardno,int axis1,int axis2,long pulse1,long pulse2)

cardno Card number

axis1,axis2 Axis number joining interpolation

pulse1,pulse2 Relative distance of movemen

Range (-8388608~+8388607)

Return 0: Correct 1: Wrong

5.5 3-axis interpolation

int inp_move3(int cardno,int axis1,int axis2,int axis3,long pulse1,long

pulse2,long pulse3)

cardno Card number

axis1 ,axis2,axis3 Axis number joining interpolation

pulse1,pulse2,pulse3 Relative distance of moiotn along specified axis (axis1/ axis2/ axis3)

Range (-8388608~+8388607)

Return 0: Correct

1: Wrong

5.6 4-axis interpolation

```
int inp_move4(int cardno,long pulse1,long pulse2,long pulse3,long pulse4)
```

cardno Card number

pulse1,pulse2,pulse3,pulse4

Relative distance of movement along X-Y-Z-W axis

Range (-8388608~+8388607)

Return 0: Correct

1: Wrong

CATEGORY OF SWITCH AMOUNT INPUT/ OUTPUT

6.1 Read single input point

```
int read_bit(int cardno,int number)
```

cardno Card number

number Input point (0-39)

Return 0: low level 1: high level -1: error

6.2 Output single output point

```
int write_bit(int cardno,int number,int value)
```

cardno Card number

number Output point (0-15)

value 0: low 1: high

Return

0: correct

1: wrong

A number corresponding to the output number

CATEGORY OF COMPOSITE DRIVING

To provide convenience for the customers, we encapsulated composite driving

functions in the basic library functions. These functions mainly integrate speed mode setting, speed parameter setting and motion functions, while absolute motion and relative motion are also considered.

7.1 Single axis symmetric relative moving

```
int symmetry_relative_move(int cardno, int axis1, long pulse1, long lspd ,long  
hspd, double tacc)
```

cardno-card number

axis1---axis number1

pulse1-- pulse of axis 1

lspd --- Low speed

hspd --- High speed

tacc--- Time of acceleration (Unit: sec)

Return 0: Correct 1: Wrong

7.2 Single axis symmetric absolute moving

```
int symmetry_absolute_ move (int cardno, int axis1, int axis2, long pulse1, long  
pulse2, long lspd ,long hspd, double tacc)
```

cardno-card number

axis1---axis number1

pulse1-- pulse of axis 1

lspd --- Low speed

hspd --- High speed

tacc--- Time of acceleration (Unit: sec)

Return 0: Correct 1: Wrong

7.3 Two axes symmetric linear interpolation relative moving

```
int symmetry_relative_line2(int cardno, int axis1, int axis2, long pulse1, long  
pulse2, long lspd ,long hspd, double tacc)
```

cardno-card number

axis1---axis number1

axis2---axis number2

pulse1-- pulse of axis 1
pulse2-- pulse of axis 2
lspd --- Low speed
hspd --- High speed
tacc--- Time of acceleration (Unit: sec)
Return 0: Correct 1: Wrong

7.4 Two axes symmetric linear interpolation absolute moving

```
int symmetry_absolute_line2(int cardno, int axis1, int axis2, long pulse1, long pulse2, long lspd ,long hspd, double tacc)
```

cardno-card number
axis1---axis number1
axis2---axis number2
pulse1-- pulse of axis 1
pulse2-- pulse of axis 2
lspd --- Low speed
hspd --- High speed
tacc--- Time of acceleration (Unit: sec)
Return 0: Correct 1: Wrong

7.5 Three axes symmetric linear interpolation relative moving

```
int symmetry_relative_line3(int cardno, int axis1, int axis2, int axis3, long pulse1, long pulse2, long pulse3, long lspd ,long hspd, double tacc)
```

cardno-card number
axis1---axis number1
axis2---axis number2
axis3---axis number3
pulse1-- pulse of axis 1
pulse2-- pulse of axis 2
pulse3-- pulse of axis 3
lspd --- Low speed

hspd --- High speed

tacc--- Time of acceleration (Unit: sec)

Return 0: Correct 1: Wrong

7.6 Three axes symmetric linear interpolation absolute moving

**int symmetry_absolute_line3(int cardno, int axis1, int axis2, int axis3, long pulse1,
long pulse2, long pulse3, long lspd ,long hspd, double tacc)**

cardno-card number

axis1---axis number1

axis2---axis number2

axis3---axis number3

pulse1-- pulse of axis 1

pulse2-- pulse of axis 2

pulse3-- pulse of axis 3

lspd --- Low speed

hspd --- High speed

tacc--- Time of acceleration (Unit: sec)

Return 0: Correct 1: Wrong

7.7 Four axes symmetric linear interpolation relative moving

**int symmetry_relative_line4(int cardno, long pulse1, long pulse2, long pulse3,
long pulse4,long lspd ,long hspd, double tacc)**

cardno-card number

pulse1-- pulse of axis 1

pulse2-- pulse of axis 2

pulse3-- pulse of axis 3

pulse4-- pulse of axis 4

lspd --- Low speed

hspd --- High speed

tacc--- Time of acceleration (Unit: sec)

Return 0: Correct 1: Wrong

7.8 Four axes symmetric linear interpolation absolute moving

```
int symmetry_absolute_line4(int cardno, long pulse1, long pulse2, long pulse3,  
long pulse4, long lspd, long hspd, double tacc);
```

cardno-- card number

pulse1-- pulse of axis 1

pulse2-- pulse of axis 2

pulse3-- pulse of axis 3

pulse4-- pulse of axis 4

lspd --- Low speed

hspd --- High speed

tacc--- Time of acceleration (Unit: sec)

Return 0: Correct 1: Wrong

CATEGORY OF EXTERNAL SIGNAL DRIVING

8.1 Quantitative drive function of external signal

```
int manual_pmove(int cardno, int axis, long pos)
```

cardno-- card number

axis --axis number

pulse-- pulse

Return 0: Correct 1: Wrong

Note: (1) Send out quantitative pulse, but the drive does not start immediately until the external signal level changes.

(2) Ordinary button and handwheel are acceptable.

8.2 Continuous drive function of external signal

```
int manual_continue(int cardno, int axis)
```

cardno-- card number

axis --axis number

Return 0: Correct 1: Wrong

Note: (1) Send out quantitative pulse, but the drive does not start immediately until the external signal level changes.

(2) Ordinary button and handwheel are acceptable.

8.3 Shut down the enabling of external signal drive

int manual_disable(int cardno, int axis)

cardno-- card number

axis --axis number

Return 0: Correct 1: Wrong

CATEGORY OF LOCK POSITION

9.1 lock the logical position and real position for all axes

int set_lock_position(int cardno, int axis,int mode,int regi,int logical)

cardno —card number

axis —reference axis

mode —set lock mode 0:inefficacy 1:efficiency

regi —register mode 0:logical position 1:real position

logical—level signal 0: from high to low 1:from low to high

Return 0: Correct 1: Wrong

9.2 Get the status of position lock

int get_lock_status(int cardno, int axis, int *v)

cardno card number

axis axis number(1-4)

status Lock status (0: unlocked, 1: locked)

Return 0: Correct 1: Wrong

9.3 Get_lock_position(int cardno,int axis,long *pos)

int get_lock_position(int cardno,int axis,long *pos)

cardno card number

axis axis number

pos lock position

Return 0: Correct 1: Wrong

9.4 Clr_lock_status(int cardno, int axis)

int clr_lock_position(int cardno,int axis,long *pos)

cardno card number

axis axis number

Return 0: Correct 1: Wrong

CATEGORY OF HARDWARE CACHE

10.1 Single axis FIFO

Int fifo_inp_move1(int cardno,int axis1,long pulse1,long speed)

cardno card number

axis1 axis number(1-4)

pulse1 pulses in FIFO cache

speed FIFO speed

Return 0: Correct 1: Wrong

10.2 Two axes FIFO

Int fifo_inp_move2(int cardno,int axis1, ,int axis2,long pulse1, long pulse2,long speed)

cardno card number

axis1 axis number(1-4)

axis2 axis number(1-4)

pulse1 pulses in FIFO buffer

pulse2 pulses in FIFO buffer

speed FIFO speed

Return 0: Correct 1: Wrong

10.3 Three axes FIFO

int fifo_inp_move3(int cardno,int axis1,int axis2,int axis3,long pulse1,long pulse2,long pulse3,long speed)

cardno card number

axis1	axis number(1-4)
axis2	axis number(1-4)
axis3	axis number(1-4)
pulse1	pulses in FIFO buffer
pulse2	pulses in FIFO buffer
pulse3	pulses in FIFO buffer
speed	FIFO speed
Return	0: Correct 1: Wrong

10.4 Four axes FIFO

int fifo_inp_move4(int cardno,long pulse1,long pulse2,long pulse3,long pulse4,long speed)

cardno	card number
axis1	axis number(1-4)
axis2	axis number(1-4)
axis3	axis number(1-4)
axis4	axis number(1-4)
pulse1	pulses in FIFO buffer
pulse2	pulses in FIFO buffer
pulse3	pulses in FIFO buffer
pulse4	pulses in FIFO buffer
speed	FIFO speed
Return	0: Correct 1: Wrong

10.5 Reset FIFO Cache

int reset_fifo(int cardno)

cardno	card number
Return	0: Correct 1: Wrong

10.6 Read FIFO cache To determine FIFO command havn't been implemented

int read_fifo_count(int cardno,int *value)

cardno	card number	
value	space(bytes) of commands that havn't been implemented	
Return	0: Correct	1: Wrong

10.7 Read FIFO cache To determine whether it is empty

int read_fifo_empty(int cardno)

cardno	card number	
Return	0: non -empty	1: empty

10.8 Read FIFO cache To determine whether it is full

int read_fifo_full(int cardno)

cardno	card number	
Return	0: non-full	1: full

Chapter 8 Guide to motion control function library

1. Introduction on ADT8940A1 function library

ADT8940A1 function library is actually the interface for users to operate the movement control card; users can control the movement control card to execute corresponding functions simply by calling interface functions.

The movement control card provides movement function library under DOS and dynamic link library under Windows; the following part will introduce the library calling method under DOS and Windows.

2. Calling dynamic link library under Windows

The dynamic link library ADT8940A1.dll under Windows is programmed in VC, applicable for general programming tools under Windows, including VB, VC, C++Builder, VB.NET, VC.NET, Delphi and group software LabVIEW.

2.1 Calling under VC

- (1) Create a new project;
- (2) Copy the ADT8940A1.lib and ADT8940A1.h files from

DevelopmentPackage/VC in the CD to the routing of the newly created item;

- (3) Under File View of the Work Area of the new item, right click mouse to select “Add Files to Project” and then in the pop-up file dialogue select the file type to be “Library Files(.lib)”, then search out “ADT8940A1.lib” and select it, finally click “OK” to finish loading of the static library;
- (4) Add #include “ADT8940A1.h” in the declaim part of the source file, header or overall header “StdAfx.h”.

After the above four steps, users can call functions in the dynamic link library.

Remark: The calling method under VC.NET is similar.

2.2 Calling under VB

- (1) Create a new project;
- (2) Copy the ADT8940A1.h file from DevelopmentPackage/VB in the CD to the routing of the newly created item;
- (3) Select the menu command Engineering/Add module and subsequently Save Current in the dialogue to search out the ADT8940A1.bas module file, finally click the Open button.

After the above three steps, users can call functions in the dynamic link library.

Remark: The calling method under VB.NET is similar.

2.3 Calling under C++Builder

- (1) Create a new project;
- (2) Copy the ADT8940A1.lib and ADT8940A1.h files from DevelopmentPackage/C++Builder in the CD to the routing of the newly created item;
- (3) Select the menu command “Project\Add to Project”, and in the pop-up dialogue select the file type to be “Library files(*.lib)”, then search out the “ADT8940A1.lib” file and click Open button;
- (4) Add #include “ADT8940A1.h” in the declaim part of the program file.

After the above four steps, users can call functions in the dynamic link library.

2.4 Calling under LabView 8

- (1) Create a new VI;
- (2) Copy the ADT8940A1.lib and ADT8940A1.dll files from DevelopmentPackage/LabVIEW in the CD to the routing of the newly created item;

- (3) Right click mouse in the blank area of the program interface to display the Function Palette, select "Select a VI.." and subsequently in the pop-up window select the ADT8940A1.lib file, finally select the required library function in the "Select the VI to Open" window and drag into the program interface.

After the above three steps, users can call functions in the dynamic link library.

3. Calling library functions under DOS

Function libraries under DOS are edited in Borland C3.1 and saved in the DevelopmentPackage/C++ (or C) folder. Library functions may be categorized into large and huge modes, applicable for standard C and Borland C3.1or above versions.

The method of calling function library with Borland C is as follows:

- (1) Under the development environment of Borland C, select the "Project\Open Project" command to create a new project;
- (2) Copy the ADT8940A1H.LIB or ADT8940A1L.LIB file and ADT8940A1.H file from DevelopmentPackage/ C (or C++) in the CD to the path of the newly created project;
- (3) Select the "Project\Add Item" command and further in the dialogue select "ADT8940A1H.LIB" or "ADT8940A1L.LIB", finally click the Add button;
- (4) Add #include "ADT8940A1.h statement in the user program file.

After the above four steps, users can call functions in the dynamic link library.

4. Returns of library functions and their meanings

To ensure users will correctly know execution of library functions, each library function in the function library after completion of execution will return to execution results of the library functions. Users, based on such Returns, can conveniently judge whether function calling has succeeded.

Except "int ADT8940A1_initial(void)" and "int read_bit(int cardno, int number)" with special Returns, other functions have only "0" and "1" as the Returns, where "0" means successful calling and "1" means failed calling.

The following list introduces meanings of function Returns.

Function name	Return	Meaning
ADT8940A1_initial	-1	No installation of service

	-2	PCI slot failure
	0	No installation of control card
	>0	Amount of control card
Read_bit	0	Low level
	1	High level
	-1	Card number or input point out of limit
Other functions	0	Correct
	1	Wrong

Remark: Return 1 means calling error, and the normal cause is wrong cardno (Card Number) or axis (Axis Number) passed during the process of calling library functions. Card number have their values starting as 0, 1, and 2, thus in case there is only one card, the card number must be 0; similarly values of axis number can only be 1, 2, 3 and 4, other values are all wrong.

Chapter 9 Briefing on motion control development

This card will encounter some problems during programming, but most problems are due to failure in understanding the methods of this control card. The following part will give explanation on some unusual and easy-to-misunderstand scenarios.

CARD INITIALIZATION

At the beginning users shall call the ADT8940A1_initial() function and ensure the ADT8940A1 card has been correctly installed, then set pulse output mode and limit switch mode. The above parameters shall be set for individual machine, and normally only one setting is required during program initialization, instead of any later setting.

Remark: Library function ADT8940A1_initial is the door to ADT8940A1 card, thus calling other functions are of sense only after successful card initialization with calling to this function.

SPEED SETTING

2.1 Constant speed motion

Parameter setting is so simple that users just set the drive speed equal to the starting speed; other parameters need no setting.

Relevant functions:

set_startv

set_speed

☛ **Remark: values used by functions will give the actual speed only after multiplying by the multiple rate.**

2.2 Interpolation speed

ADT8940A1 card can take any 2 axes, any 3 axes or all the 4 axes for linear interpolation.

For speed of interpolation, speed parameter for the earliest axis will apply as speed of the long axis, for example,

inp_move2 (0,3,1,100,200) is to apply the speed parameters of the first axis, i.e., X axis, independent of parameter sequence.

inp_move3 (0,3,4,2,100,200,500) is to apply the speed parameters of the second axis, i.e., Y axis, independent of parameter sequence.

Remark: speed multiple rate during interpolation is half of that during single-axis movement, which means under the same parameters speed of interpolation is only half of that of single-axis movement.

☛ STOP0, STOP1 signal

Every axis has STOP0, STOP1 .therefore, there are 8 STOP signals totally. These signals are mainly used in back-to-home operation. The back-to-home mode can use either one signal or several signals. Please note that this signal is decelerated stop. For high speed resetting, you can add one deceleration switch before home switch, i.e. use two STOP signals (one for home switch and the other for deceleration switch). You can also use one signal only. In this case, when the machine receives STOP signal, it stops in deceleration, then, moves to opposite direction in constant speed and stops when receives the signal again

Chapter 10 Programming samples in motion control development

All movement control functions return immediately; once a drive command is made, the movement process will be controlled by the movement control card until completion; then the host computer software of users can real-time monitor the whole movement process or force to stop the process.

Remark: Axis during motion are not allowed to send new drive commands to motion axis, otherwise the previous drive will be given up so as to execute the new drive.

Although programming languages vary in types, they can still be concluded as Three Structures and One Spirit. Three Structures refer to sequential structure, cycling structure and branch structure emphasized by all the programming languages, and One Spirit refer to calculation and module division involved in order to complete design assignments, which is also the key and hard point in whole programming design.

To ensure a program is popular, standard, expandable and easy for maintenance, all the later samples will be divided into the following modules in terms of project design: movement control module (to further seal library functions provided by the control card), function realization module (to cooperate code phase of specific techniques), monitoring module and stop processing module.

Now let's brief application of ADT8940A1 card function library in VB and VC; users using other programming languages may take reference.

VB PROGRAMMING SAMPLES

1.1 PREPARATION

(1) Create a new item and save as "test.vbp";

- (2) Add the ADT8940A1.bas module in the item following the above-introduced method;

1.2 Movement control module

- (1) Add a new module in the project and save as “ctrlcard.bas”;
- (2) At first, within the motion control module self-define initialization functions of the motion control card and initialize library functions to be sealed into initialization functions;
- (3) Further self-define relevant motion control functions such as speed setting function, single-axis motion function, and interpolation function;
- (4) Source code of ctrcard.bas is:

```
'***** Motion control module *****  
'  
' For developing an application system of great generality,  
' extensibility and convenient maintenance easily and swiftly,  
' we envelop all the library functions by category basing on  
' the card function library  
'*****/  
  
Public Result As Integer 'return  
Const MAXAXIS = 4 'axis number  
'*****initial motion-card*****  
'  
' this function is boot of using motion-card  
' Return<=0 fail to initial motion-card,  
' Return>0 Succeed in initial motion-card  
'*****  
  
Public Function Init_Card() As Integer  
    Result = adt8940a1_initial 'initial motion-card  
    If Result <= 0 Then  
        Init_Card = Result  
        Exit Function  
    End If  
    For i = 1 To MAXAXIS  
        set_command_pos 0, i, 0 'set logic pos as 0  
        set_actual_pos 0, i, 0 'set real pos as 0  
        set_startv 0, i, 1000 'set start-speed  
        set_speed 0, i, 2000 'set motion-speed
```

```
set_acc 0, i, 625          'set acceleration
Next i
Init_Card = Result
End Function

'*****get version*****
'   get library version and hardware version
'   para: libVer—library version,hardwareVer - hardware version
'*****

Public Function Get_Version(libver As Double, hardwarever As Double) As Integer
    Dim ver As Integer
    ver = get_lib_version(0)
    libver = (ver)
    hardwarever = get_hardware_ver(0)
End Function

'*****set speed*****
'   according as para,judge whether is constant-speed
'   set start-speed ,motion-speed and acceleration
'   para: axis    -axis number
'          startv  -start - speed
'          speed   -motion - speed
'          Add     -acceleration
'   Return=0 correct, Return=1 wrong
'*****

Public Function Setup_Speed(ByVal axis As Integer, ByVal startv As Long, ByVal
speed As Long, ByVal add As Long, ByVal tacc As Double) As Integer
    If (startv - speed >= 0) Then
        Result = set_startv(0, axis, startv)
        set_speed 0, axis, startv
    '    set_symmetry_speed 0, axis, startv, startv, tacc
    Else
        Result = set_startv(0, axis, startv)
        set_speed 0, axis, speed
```

```
        set_acc 0, axis, add / 125
    '
        set_symmetry_speed 0, axis, startv, speed, tacc
    End If
End Function
```

```
/'*****single-axis motion*****'
```

```
'    drive one axis motion
'    para:  axis-axis number,  value-pulse of motion
'    Return=0 correct,  Return=1 wrong
```

```
*****/
```

```
Public Function Axis_Pmove(ByVal axis As Integer, ByVal pulse As Long) As Integer
```

```
    Result = pmove(0, axis, pulse)
```

```
    Axis_Pmove = Result
```

```
End Function
```

```
/'*****2-axis interpolation*****'
```

```
'    any 2-axis linear interpolation
'    para:  axis1,axis2-axis number、 value1,value2-pulse of interpolation
'    Return=0 correct,  Return=1 wrong
```

```
*****/
```

```
Public Function Interp_Move2(ByVal axis1 As Integer, ByVal axis2 As Integer, ByVal pulse1 As Long, ByVal pulse2 As Long) As Integer
```

```
    Result = inp_move2(0, axis1, axis2, pulse1, pulse2)
```

```
    Interp_Move2 = Result
```

```
End Function
```

```
/'*****3-axis interpolation*****'
```

```
'    any 3-axis linear interpolation
'    para :  axis1,axis2,axis3-axis number 、 value1,value2,value3-pulse of interpolation
'    Return=0 correct,  Return=1 wrong
```

```
*****/
```

```
Public Function Interp_Move3(ByVal axis1 As Integer, ByVal axis2 As Integer, ByVal axis3 As Integer, ByVal pulse1 As Long, ByVal pulse2 As Long, ByVal pulse3 As Long) As Integer
```

pulse3 As Long) As Integer

Result = inp_move3(0, axis1, axis2, axis3, pulse1, pulse2, pulse3)

Interp_Move3 = Result

End Function

*/*****4-axis interpolation*****/

' 4-axis interpolation motion

' para: value1,value2,value3,value4-pulse of interpolation

' Return=0 correct, Return=1 wrong

*/*****/

Public Function Interp_Move4(ByVal pulse1 As Long, ByVal pulse2 As Long,
ByVal pulse3 As Long, ByVal pulse4 As Long) As Integer

Result = inp_move4(0, pulse1, pulse2, pulse3, pulse4)

Interp_Move4 = Result

End Function

*/*****stop motion*****/

' stop motion in the way of sudden or decelerate

' para: axis-axis number、mode-stop mode(0—sudden stop, 1—decelerate
stop)

' Return=0 correct, Return=1 wrong

*/*****/

Public Function StopRun(ByVal axis As Integer, ByVal mode As Integer) As Integer

If mode = 0 Then

Result = sudden_stop(0, axis)

Else

Result = dec_stop(0, axis)

End If

End Function

*/*****set position counter*****/

' set logic-pos or real-pos

' para: axis-axis number,pos-the set value

' mode 0—set logic pos,non 0—set real pos

```
'      Return=0 correct,  Return=1 wrong
*****/

Public Function Setup_Pos(ByVal axis As Integer, ByVal pos As Long, ByVal mode
As Integer) As Integer
    If mode = 0 Then
        Result = set_command_pos(0, axis, pos)
    Else
        Result = set_actual_pos(0, axis, pos)
    End If
End Function

'*****get information of motion*****
'      get logical-pos,actual-pos and motion-speed
'      para:  axis-axis number,LogPos-logic pos,ActPos-real pos,Speed-motion
speed
'      Return=0 correct,  Return=1 wrong
*****/

Public Function Get_CurrentInf(ByVal axis As Integer, LogPos As Long, actpos As
Long, speed As Long) As Integer
    Result = get_command_pos(0, axis, LogPos)
    get_actual_pos 0, axis, actpos
    get_speed 0, axis, speed
    Get_CurrentInf = Result
End Function

'*****get status of motion*****
'      get status of single-axis motion or interpolation
'      para :  axis-axis number ,  value-Indicator of motion status(0 : Drive
completed,Non-0: Drive in process)
'      mode(0-single-axis motion,  1—interpolation)
'      Return=0 correct,  Return=1 wrong
*****/

Public Function Get_MoveStatus(ByVal axis As Integer, value As Long, ByVal
mode As Integer) As Integer
```



```
If mode = 0 Then
    GetMove_Status = get_status(0, axis, value)
Else
    GetMove_Status = get_inp_status(0, value)
End If
End Function
```

```
/'*****read input*****'
'    read status of input
'    para: number-input port(0 ~ 39)
'    Return: 0 — low level, 1 — high level, -1 — error
'*****/'
```

```
Public Function Read_Input(ByVal number As Integer) As Integer
    Read_Input = read_bit(0, number)
End Function
```

```
/'*****output*****'
'    set status of output
'    para: number-output port(0 ~ 15),value 0-low level、 1—high level
'    Return=0 correct, Return=1 wrong
'*****/'
```

```
Public Function Write_Output(ByVal number As Integer, ByVal value As Integer)
As Integer
    Write_Output = write_bit(0, number, value)
End Function
```

```
/'*****set pulse mode*****'
'    set the mode of pulse output
'    para: axis-axis number, value-pulse mode 0—pulse+pulse 1—pulse +
direction
'    Return=0 correct, Return=1 wrong
'    Default mode: Pulse + direction, with positive logic pulse
'    and positive logic direction input signal
'
```

*****/

Public Function Setup_PulseMode(ByVal axis As Integer, ByVal value As Integer)
As Integer

 Setup_PulseMode = set_pulse_mode(0, axis, value, 0, 0)

End Function

/'*****set nLMT mode*****/

' set the mode of nLMT signal input along positive/ negative direction

' para: axis—axis number

' value1 0 - positive limit effective 1: positive limit ineffective

' value2 0: negative limit effective 1: negative limit ineffective

' logic 0: low level effective 1: high level ineffective

' Default mode: Apply positive and negative limits with low level

' Return 0: Correct 1: Wrong

' *****/

Public Function Setup_LimitMode(ByVal axis As Integer, ByVal value1 As Integer,
ByVal value2 As Integer, ByVal logic As Integer) As Integer

 Setup_LimitMode = set_limit_mode(0, axis, value1, value2, logic)

End Function

/'*****set stop0 mode*****/

' Set mode of stop0 input signal

' para: axis—axis number

' value 0—ineffective 1—effective

' logic 0—low level effective 1—high level effective

' Defaule: ineffective

' Return 0: Correct 1: Wrong

' *****/

Public Function Setup_Stop0Mode(ByVal axis As Integer, ByVal value As Integer,
ByVal logic As Integer) As Integer

 Setup_Stop0Mode = set_stop0_mode(0, axis, value, logic)

End Function

/'*****set stop1 mode*****/

```
' Set mode of stop1 input signal
' para: axis—axis number
'       value  0—ineffective  1—effective
'       logic   0—low level effective  1—high level effective
' Defaule:      ineffective
' Return  0: Correct                      1: Wrong
' *****/
```

Public Function Setup_Stop1Mode(ByVal axis As Integer, ByVal value As Integer, ByVal logic As Integer) As Integer

Setup_Stop1Mode = set_stop1_mode(0, axis, value, logic)

End Function

```
/'*****set hardware-stop mode *****
' set mode whether Hardware stop is effective,if hareware-version is 1 ,
' motion-card havn't this function
' para: value  0—ineffective  1—effective
'       logic   0—low level effective  1—high level effective
' Defaule:      ineffective
' Return  0: Correct                      1: Wrong
' Hardware stop signals are assigned to use the 34 pin at the P3 terminal panel
(IN31)
' *****/
```

Public Function Setup_HardStop(ByVal value As Integer, ByVal logic As Integer) As Integer

Setup_HardStop = set_suddenstop_mode(0, value, logic)

End Function

```
/'*****set delay*****
' set the time of delay,if hareware-version is 1 ,motion-card havn't this function
' para: time - time of delay(unit is us)
' Return  0: Correct                      1: Wrong
' *****/
```

Public Function Setup_Delay(ByVal time As Long) As Integer

Setup_Delay = set_delay_time(0, time * 8)

End Function

```
'/*****get delay status *****/
```

```
'  get the status of delay ,if haware-version is 1 ,motion-card havn't this  
function
```

```
'  Return    0: delay stop          1: delay in process
```

```
'  *****/
```

Public Function Get_DelayStatus() As Integer

 Get_DelayStatus = get_delay_status(0)

End Function

```
'/***** Symmetrical relative movement of single-axis *****/
```

```
'function: Refer to the current position and perform quantitative movement in the  
symmetrical
```

```
'acceleration/deceleration
```

```
'para:
```

```
'    axis---axis number
```

```
'    pulse --pulse
```

```
'    lspd--- Low speed
```

```
'    hspd--- High speed
```

```
'    tacc--- Time of acceleration (Unit: sec)
```

```
'return value 0: correct 1: wrong
```

```
*****/
```

Public Function Sym_RelativeMove(ByVal axis As Integer, ByVal pulse As Long,
ByVal lspd As Long, ByVal hspd As Long, ByVal tacc As Double) As Integer

 Result = symmetry_relative_move(0, axis, pulse, lspd, hspd, tacc)

 Symmetry_RelativeMove = Result

End Function

```
'/***** Symmetrical absolute movement of single-axis *****/
```

```
'function: Refer to the position of zero point and perform quantitative movement in  
the symmetrical
```

```
'acceleration/deceleration
```

```
'para:
```

```
'    axis ---axis number
```

```
'    pulse --pulse
```

```
'    lspd --- Low speed
'    hspd --- High speed
'    tacc--- Time of acceleration (Unit: sec)
```

```
'return value 0: correct 1: wrong
```

```
*****/
```

```
Public Function Sym_AbsoluteMove(ByVal axis As Integer, ByVal pulse As Long,
ByVal lspd As Long, ByVal hspd As Long, ByVal tacc As Double) As Integer
```

```
    Result = symmetry_absolute_move(0, axis, pulse, lspd, hspd, tacc)
```

```
    Symmetry_AbsoluteMove = Result
```

```
End Function
```

```
/'***** Relative movement of two-axis symmetrical linear interpolation *****
```

```
'function: Refer to current position and perform linear interpolation in symmetrical
```

```
'acceleration/deceleration
```

```
'para:
```

```
'    axis1---axis number1
'    axis2---axis number2
'    pulse1-- pulse 1
'    pulse2-- pulse 2
'    lspd --- Low speed
'    hspd --- High speed
'    tacc--- Time of acceleration (Unit: sec)
```

```
'return value 0: correct 1: wrong
```

```
*****/
```

```
Public Function Sym_RelativeLine2(ByVal axis1 As Integer, ByVal axis2 As
Integer, ByVal pulse1 As Long, ByVal pulse2 As Long, ByVal lspd As Long, ByVal
hspd As Long, ByVal tacc As Double) As Integer
```

```
    Result = symmetry_relative_line2(0, axis1, axis2, pulse1, pulse2, lspd, hspd,
tacc)
```

```
    Symmetry_RelativeLine2 = Result
```

```
End Function
```

```
/'***** Two axes symmetric linear interpolation absolute moving *****
```

```
'function: Refer to the position of zero point and perform linear interpolation in
```

symmetrical

'acceleration/deceleration

'para:

' axis1---axis number1
' axis2---axis number2
' pulse1---pulse of axis 1
' pulse2-- pulse of axis 2
' lspd --- Low speed
' hspd --- High speed
' tacc--- Time of acceleration (Unit: sec)

'return value 0: correct 1: wrong

*****/

Public Function Sym_AbsoluteLine2(ByVal axis1 As Integer, ByVal axis2 As Integer, ByVal pulse1 As Long, ByVal pulse2 As Long, ByVal lspd As Long, ByVal hspd As Long, ByVal tacc As Double) As Integer

Result = symmetry_absolute_line2(0, axis1, axis2, pulse1, pulse2, lspd, hspd, tacc)

Symmetry_AbsoluteLine2 = Result

End Function

/'**** Three axes symmetric linear interpolation relative moving ****'

'function: Refer to current position and perform linear interpolation in symmetric

'acceleration/deceleration

'para:

' axis1---axis number1
' axis2---axis number2
' axis3---axis number3
' pulse1-- pulse of axis 1
' pulse2-- pulse of axis 2
' pulse3-- pulse of axis 3
' lspd --- Low speed
' hspd --- High speed
' tacc--- Time of acceleration (Unit: sec)

'return value 0: correct 1: wrong

*****/

Public Function Sym_RelativeLine3(ByVal axis1 As Integer, ByVal axis2 As Integer, ByVal axis3 As Integer, ByVal pulse1 As Long, ByVal pulse2 As Long, ByVal pulse3 As Long, ByVal lspd As Long, ByVal hspd As Long, ByVal tacc As Double) As Integer

Result = symmetry_relative_line3(0, axis1, axis2, axis3, pulse1, pulse2, pulse3, lspd, hspd, tacc)

Symmetry_RelativeLine3 = Result

End Function

'*****Three axes symmetric linear interpolation absolute moving *****

'function: Refer to the position of zero point and perform linear interpolation in symmetric

'acceleration/deceleration.

'para:

' axis1---axis number1
' axis2---axis number2
' axis3---axis number3
' pulse1-- pulse of axis 1
' pulse2-- pulse of axis 2
' pulse3-- pulse of axis 3
' lspd --- Low speed
' hspd --- High speed
' tacc--- Time of acceleration (Unit: sec)

'return value 0: correct 1: wrong

*****/

Public Function Sym_AbsoluteLine3(ByVal axis1 As Integer, ByVal axis2 As Integer, ByVal axis3 As Integer, ByVal pulse1 As Long, ByVal pulse2 As Long, ByVal pulse3 As Long, ByVal lspd As Long, ByVal hspd As Long, ByVal tacc As Double) As Integer

Result = symmetry_absolute_line3(0, axis1, axis2, axis3, pulse1, pulse2, pulse3, lspd, hspd, tacc)

Symmetry_AbsoluteLine3 = Result

End Function

'***** Four axes symmetric linear interpolation relative moving *****

'function: Refer to current position and perform linear interpolation in symmetric

'acceleration/deceleration

'para:

' pulse1-- pulse of axis 1

' pulse2-- pulse of axis 2

' pulse3-- pulse of axis 3

' pulse4-- pulse of axis 4

' lspd --- Low speed

' hspd --- High speed

' tacc--- Time of acceleration (Unit: sec)

'return value 0: correct 1: wrong

*****/

Public Function Sym_RelativeLine4(ByVal pulse1 As Long, ByVal pulse2 As Long,
ByVal pulse3 As Long, ByVal pulse4 As Long, ByVal lspd As Long, ByVal hspd As
Long, ByVal tacc As Double) As Integer

Result = symmetry_relative_line4(0, pulse1, pulse2, pulse3, pulse4, lspd,
hspd, tacc)

Symmetry_RelativeLine4 = Result

End Function

'*****Four axes symmetric linear interpolation absolute moving *****

'function: Refer to the position of zero point and perform linear interpolation in
symmetric

'acceleration/deceleration.

'para:

' pulse1-- pulse of axis 1

' pulse2-- pulse of axis 2

' pulse3-- pulse of axis 3

' pulse4-- pulse of axis 4


```
'      lspd --- Low speed
'      hspd --- High speed
'      tacc--- Time of acceleration (Unit: sec)
'return value 0: correct 1: wrong
'*****/

Public Function Sym_AbsoluteLine4(ByVal pulse1 As Long, ByVal pulse2 As Long,
ByVal pulse3 As Long, ByVal pulse4 As Long, ByVal lspd As Long, ByVal hspd As
Long, ByVal tacc As Double) As Integer
    Result = symmetry_absolute_line4(0, pulse1, pulse2, pulse3, pulse4, lspd,
hspd, tacc)
    Symmetry_AbsoluteLine4 = Result
End Function

'***** Quantitative drive function of external signal *****
'function: Quantitative drive function of external signal
'para:
'      cardno    card number
'      axis      axis number
'      pulse pulse
'Return 0: Correct 1: Wrong
'*****/

Public Function Manu_Pmove(ByVal axis As Integer, ByVal pulse As Long) As
Integer
    Result = manual_pmove(0, axis, pulse)
    Manu_Pmove = Result
End Function

'***** Continuous drive function of external signal *****
'function: Continuous drive function of external signal
'para:
'      cardno    card number
'      axis      axis number
'Return 0: Correct 1: Wrong
'*****/
```

Public Function Manu_Continue(ByVal axis As Integer) As Integer

Result = manual_continue(0, axis)

Manu_Continue = Result

End Function

/'***** Shut down the enabling of external signal drive *****

'function: Shut down the enabling of external signal drive

'para:

' cardno card number

' axis axis number

'Return 0: Correct 1: Wrong

*****/

Public Function Disable_Manu(ByVal axis As Integer) As Integer

Result = manual_disable(0, axis)

Disable_Manu = Result

End Function

/'***** set lockmode *****

'function:lock the logical position and real position for all axis

'para:

' axis—reference axis

' mode--set lock mode |0:inefficacy

' |1:efficiency

' regi—register mode |0:logical position

' |1:real position

' logical—level signal |0: from high to low

' |1:from low to high

'retutrn 0: correct 1: wrong

'Note: Use IN signal of specific axis as the trigger signal

*****/

Public Function Get_LockStatus(ByVal axis As Integer, status As Integer) As Integer

Result = get_lock_status(0, axis, status)

Get_LockStatus = Result

End Function

```
/'***** get synchronous action state *****'
```

```
'function:get synchronous action state
```

```
'para:
```

```
'    axis      axis number
```

```
'    status—  0|haven't run synchronous
```

```
'              1|run synchronous
```

```
'return 0: correct 1: wrong
```

```
'Note: This function could tell whether the position lock has been executed
```

```
/'*****/'
```

```
Public Function Setup_LockPosition(ByVal axis As Integer, ByVal mode As Integer,
```

```
ByVal regi As Integer, ByVal logical As Integer) As Integer
```

```
    Result = set_lock_position(0, axis, mode, regi, logical)
```

```
    Setup_LockPosition = Result
```

End Function

```
/'*****get lock position*****'
```

```
'Function: Get the locked position
```

```
'para:
```

```
'    axis      axis number
```

```
'    pos lock position
```

```
'Return 0: Correct 1: Wrong
```

```
/'*****/'
```

```
Public Function Get_LockPosition(ByVal axis As Integer, pos As Long) As Integer
```

```
    Result = get_lock_position(0, axis, pos)
```

```
    Get_LockPosition = Result
```

End Function

```
/'*****clean lock position*****'
```

```
'Function: Clean the locked position
```

```
'para:
```

```
'    axis      axis number
```

```
'Return 0: Correct 1: Wrong
```

*****/

Public Function Clr_LockStatus(ByVal axis As Integer) As Integer

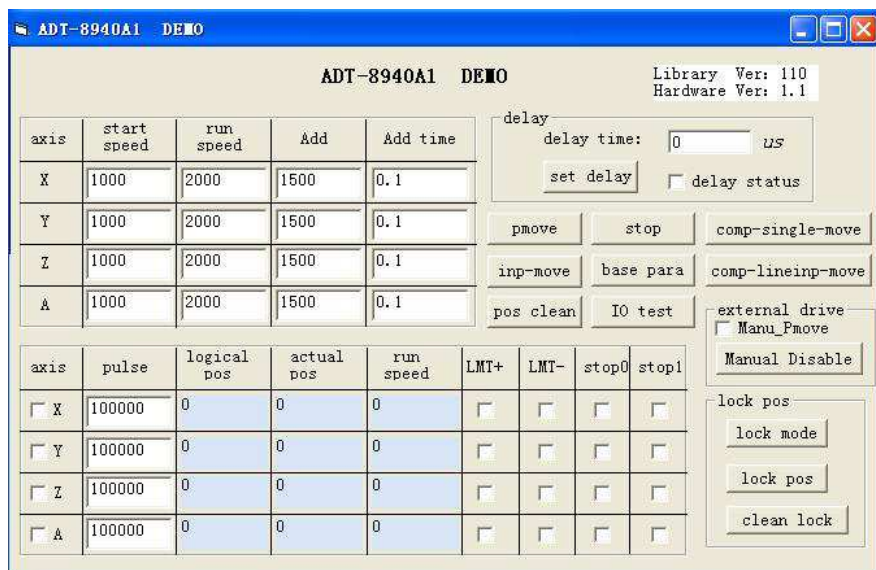
Result = clr_lock_status(0, axis)

Clr_LockStatus = Result

End Function

1.3 Function realization module

1.31 Interface design



axis	start speed	run speed	Add	Add time
X	1000	2000	1500	0.1
Y	1000	2000	1500	0.1
Z	1000	2000	1500	0.1
A	1000	2000	1500	0.1

axis	pulse	logical pos	actual pos	run speed	LMT+	LMT-	stop0	stop1
☐ X	100000	0	0	0	☐	☐	☐	☐
☐ Y	100000	0	0	0	☐	☐	☐	☐
☐ Z	100000	0	0	0	☐	☐	☐	☐
☐ A	100000	0	0	0	☐	☐	☐	☐

Introduction:

- (1) Speed setting part—used to set starting speed, motion speed and acceleration of every axis; position setting—used to set drive pulse for every axis; drive information—used to real-time display logical position, real position and operation speed of every axis;
- (2) Motion object—users determine axis joining simultaneous motion or interpolation by selecting drive objects;
- (3) Simultaneous movement—Used to send single-axis drive commands to all the axis of the selected drive object; interpolation —Used to send interpolation command to all the axis of the selected drive object; stop—stop all the pulse

outputs of all axis.

All the above data take pulse as the unit.

1.3.2 Initialization codes are inside the window loading event, with the following contents:

```
Private Sub Init_Board()  
    Dim count As Integer  
    count = Init_Card  
    If count < 1 Then MsgBox "Fail to initial ADT-8940A1 motion-card"  
    Get_Version g_nLibVer, g_nHardwareVer  
    CardVer.Caption = "Library  Ver:  " + CStr(g_nLibVer) + Chr(10) + "Hardware Ver:  " +  
        CStr(g_nHardwareVer)  
End Sub
```

1.3.3 Simultaneous movement codes are inside the click event of axisPmove button, whereby various selected objects send corresponding drive commands. The four check boxes (to select objects) are respectively named as X, Y, Z and A, subject with the following code:

```
Private Sub AxisPmove_Click()  
*****judge speed whether is out of range*****  
' The range of start-speed and run-speed (1~2M)  
' The range of add( $1 \times 125 \sim 64000 \times 125$ )  
*****  
    If m_bX.value = vbChecked Then  
        Setup_Speed 1, m_nStartV(0).Text, m_nSpeed(0).Text, m_nAdd(0).Text,  
m_dTacc(0).Text  
        Axis_Pmove 1, m_nPulse(0).Text  
    End If  
    If m_bY.value = vbChecked Then  
        Setup_Speed 2, m_nStartV(1).Text, m_nSpeed(1).Text, m_nAdd(1).Text,  
m_dTacc(1).Text  
        Axis_Pmove 2, m_nPulse(1).Text  
    End If
```

```
If m_bZ.value = vbChecked Then
    Setup_Speed 3, m_nStartV(2).Text, m_nSpeed(2).Text, m_nAdd(2).Text,
m_dTacc(2).Text
    Axis_Pmove 3, m_nPulse(2).Text
End If
If m_bA.value = vbChecked Then
    Setup_Speed 4, m_nStartV(3).Text, m_nSpeed(3).Text, m_nAdd(3).Text,
m_dTacc(3).Text
    Axis_Pmove 4, m_nPulse(3).Text
End If
End Sub
```

1.3.4 Interpolation codes are inside the click event of InterpMove button, whereby various selected objects send corresponding drive commands. The four check boxes (to select objects) are respectively named as X, Y, Z and A, subject with the following code:

```
Private Sub InterpMove_Click()
```

```
'*****judge speed whether is out of range*****'
```

```
' The range of start-speed and run-speed(1~2M)
```

```
' The range of add (1×125~64000×125)
```

```
'*****'
```

```
'*****inptpolation*****'
```

```
'*****4-axis
```

```
linear-inptpolation*****'
```

```
If m_bX.value = vbChecked And m_bY.value = vbChecked And m_bZ.value = vbChecked
And m_bA.value = vbChecked Then
```

```
    Setup_Speed 1, m_nStartV(0).Text, m_nSpeed(0).Text, m_nAdd(0).Text,
m_dTacc(0).Text
```

```
    Interp_Move4 m_nPulse(0).Text, m_nPulse(1).Text, m_nPulse(2).Text,
m_nPulse(3).Text
```

```
'*****3-axis
```

```
linear-inptpolation*****'
```

```
'*****XYZ*****'
```

```
Elseif m_bX.value = vbChecked And m_bY.value = vbChecked And m_bZ.value =
vbChecked Then
```

```
Setup_Speed 1, m_nStartV(0).Text, m_nSpeed(0).Text, m_nAdd(0).Text,
m_dTacc(0).Text
Interp_Move3 1, 2, 3, m_nPulse(0).Text, m_nPulse(1).Text, m_nPulse(2).Text
'*****XYW*****
ElseIf m_bX.value = vbChecked And m_bY.value = vbChecked And m_bA.value =
vbChecked Then
Setup_Speed 1, m_nStartV(0).Text, m_nSpeed(0).Text, m_nAdd(0).Text,
m_dTacc(0).Text
Interp_Move3 1, 2, 4, m_nPulse(0).Text, m_nPulse(1).Text, m_nPulse(3).Text
'*****XZW*****
ElseIf m_bX.value = vbChecked And m_bZ.value = vbChecked And m_bA.value =
vbChecked Then
Setup_Speed 1, m_nStartV(0).Text, m_nSpeed(0).Text, m_nAdd(0).Text,
m_dTacc(0).Text
Interp_Move3 1, 3, 4, m_nPulse(0).Text, m_nPulse(2).Text, m_nPulse(3).Text
'*****YZW*****
ElseIf m_bY.value = vbChecked And m_bZ.value = vbChecked And m_bA.value =
vbChecked Then
Setup_Speed 2, m_nStartV(1).Text, m_nSpeed(1).Text, m_nAdd(1).Text,
m_dTacc(1).Text
Interp_Move3 2, 3, 4, m_nPulse(1).Text, m_nPulse(2).Text, m_nPulse(3).Text
'*****2-axis
linear-interpolation*****
'*****XY*****
ElseIf m_bX.value = vbChecked And m_bY.value = vbChecked Then
Setup_Speed 1, m_nStartV(0).Text, m_nSpeed(0).Text, m_nAdd(0).Text,
m_dTacc(0).Text
Interp_Move2 1, 2, m_nPulse(0).Text, m_nPulse(1).Text
'*****XZ*****
ElseIf m_bX.value = vbChecked And m_bZ.value = vbChecked Then
Setup_Speed 1, m_nStartV(0).Text, m_nSpeed(0).Text, m_nAdd(0).Text,
m_dTacc(0).Text
Interp_Move2 1, 3, m_nPulse(0).Text, m_nPulse(2).Text
'*****XW*****
ElseIf m_bX.value = vbChecked And m_bA.value = vbChecked Then
Setup_Speed 1, m_nStartV(0).Text, m_nSpeed(0).Text, m_nAdd(0).Text,
m_dTacc(0).Text
Interp_Move2 1, 4, m_nPulse(0).Text, m_nPulse(3).Text
'*****YZ*****
ElseIf m_bY.value = vbChecked And m_bZ.value = vbChecked Then
```

```
        Setup_Speed    2,    m_nStartV(1).Text,    m_nSpeed(1).Text,    m_nAdd(1).Text,
m_dTacc(1).Text
        Interp_Move2 2, 3, m_nPulse(1).Text, m_nPulse(2).Text
'*****YW*****
        ElseIf m_bY.value = vbChecked And m_bA.value = vbChecked Then
            Setup_Speed    2,    m_nStartV(1).Text,    m_nSpeed(1).Text,    m_nAdd(1).Text,
m_dTacc(1).Text
            Interp_Move2 2, 4, m_nPulse(1).Text, m_nPulse(3).Text
'*****ZW*****
            ElseIf m_bZ.value = vbChecked And m_bA.value = vbChecked Then
                Setup_Speed    3,    m_nStartV(2).Text,    m_nSpeed(2).Text,    m_nAdd(2).Text,
m_dTacc(2).Text
                Interp_Move2 3, 4, m_nPulse(2).Text, m_nPulse(3).Text
            Else
                MsgBox "please choose the axis", , "Notice"
            End If
        End Sub
End Sub
```

1.4 Monitoring module

The monitoring module is used to real-time get motion information of all the axes and display motion information, at the same time of controlling them in motion process without any new motion commands. This module is completed by the timer event, with the following codes:

```
Private Sub Timer1_Timer()
    Dim nLogPos As Long           'logic pos
    Dim nActPos As Long           'real pos
    Dim nSpeed As Long            'run-speed
    Dim nStatus(4) As Long        'status of motion
    For i = 1 To 4
        Get_CurrentInf i, nLogPos, nActPos, nSpeed
        m_nLogPos(i - 1).Caption = nLogPos
        m_nActPos(i - 1).Caption = nActPos
        m_nRunSpeed(i - 1).Caption = nSpeed
        Get_MoveStatus i, nStatus(i - 1), 0
    'Check signal of limit、stop0 and stop1
    'LMT+(XLMT-: 0,YLMT-:6,ZLMT-:12,WLMT-:18)
        If Read_Input((i - 1) * 6) = 0 Then
            m_bPLimit(i - 1).value = 1
        End If
    Next i
End Sub
```



```
Else
    m_bPLimit(i - 1).value = 0
End If
'LMT-(XLMT+ : 1,YLMT+ :7,ZLMT+ :13,WLMT+ :19)
If Read_Input((i - 1) * 6 + 1) = 0 Then
    m_bNLimit(i - 1).value = 1
Else
    m_bNLimit(i - 1).value = 0
End If
'stop0(XSTOP0 : 2,YSTOP0 :8,ZSTOP0 :14,WSTOP0 :20)
If Read_Input((i - 1) * 6 + 2) = 0 Then
    m_bStop0(i - 1).value = 1
Else
    m_bStop0(i - 1).value = 0
End If
'stop1(XSTOP1 : 3,YSTOP1 :9,ZSTOP1 :15,WSTOP1 :21)
If Read_Input((i - 1) * 6 + 3) = 0 Then
    m_bStop1(i - 1).value = 1
Else
    m_bStop1(i - 1).value = 0
End If
Next i
If nStatus(0) = 0 And nStatus(1) = 0 And nStatus(2) = 0 And nStatus(3) = 0 Then
    'is running
    AxisPmove.Enabled = True
    InterpMove.Enabled = True
    BaseparaSet.Enabled = True
    ClearPos.Enabled = True
    ComeMove.Enabled = True
    LineInpMove.Enabled = True
    IOTest.Enabled = True
Else
    'motion is finished or stopped
    AxisPmove.Enabled = False
    InterpMove.Enabled = False
    BaseparaSet.Enabled = False
    ClearPos.Enabled = False
    ComeMove.Enabled = False
    LineInpMove.Enabled = False
    IOTest.Enabled = False
```

End If

End Sub

1.5 Stop module

This module is mainly used to control unexpected events during drive process and will immediately stop drive of all the axes. Codes of this stop module are within the click event of CmdStop button, with the following codes:

```
Private Sub CmdStop_Click()
```

```
    For i = 1 To 4
```

```
        StopRun i, 0
```

```
    Next i
```

```
End Sub
```

🔧 VC PROGRAMMING SAMPLES

2.1 Preparation

- (1) Create a new item and save as "VCExample.dsw";
- (2) Load the static library ADT8940A1.lib into the item following the above-introduced method;

2.2 Movement control module

- (1) Add a new category in the item and save the header as "CtrlCard.h" and source file as "CtrlCard.cpp";
- (2) At first, within the movement control module self-define initialization functions of the movement control card and initialize library functions to be sealed into initialization functions;
- (3) Further self-define relevant movement control functions such as speed setting function, single-axis motion function, and interpolation function;
- (4) Source codes of the header CtrlCard.h are as follows:

```
#ifndef __ADT8940A1__CARD__
```

```
#define __ADT8940A1__CARD__
```

```
/****** Motion control module *****/
```

For developing an application system of great generality,
extensibility and convenient maintenance easily and swiftly,
we envelop all the library functions by category basing on
the card function library

*****/

```
#define MAXAXIS 4 //axis number
```

```
class CCtrlCard
```

```
{  
public:  
    int Get_DelayStatus();  
    int Setup_Delay(long time);  
    int Setup_HardStop(int value, int logic);  
    int Setup_Stop1Mode(int axis, int value, int logic);  
    int Setup_Stop0Mode(int axis, int value, int logic);  
    int Setup_LimitMode(int axis, int value1, int value2, int logic);  
    int Setup_PulseMode(int axis, int value);  
    void Get_Version(float &LibVer, float &HardwareVer);  
    int Setup_Pos(int axis, long pos, int mode);  
    int Write_Output(int number, int value);  
    int Read_Input(int number);  
    int Get_CurrentInf(int axis, long &LogPos, long &ActPos, long &Speed);  
    int Get_Status(int axis, int &value, int mode);  
    int StopRun(int axis, int mode);  
    int Interp_Move4(long value1, long value2, long value3, long value4);  
    int Interp_Move3(int axis1, int axis2, int axis3, long value1, long value2, long value3);  
    int Interp_Move2(int axis1, int axis2, long value1, long value2);  
    int Axis_Pmove(int axis, long value);  
    int Setup_Speed(int axis, long startv, long speed, long add);  
    int Init_Board();  
  
    int Sym_RelativeMove(int axis, long pulse, long lspd, long hspd, double tacc);  
    int Sym_AbsoluteMove(int axis, long pulse, long lspd, long hspd, double tacc);  
    int Sym_RelativeLine2(int axis1, int axis2, long pulse1, long pulse2, long lspd, long hspd,  
double tacc);  
    int Sym_AbsoluteLine2(int axis1, int axis2, long pulse1, long pulse2, long lspd, long hspd,  
double tacc);  
    int Sym_RelativeLine3(int axis1, int axis2, int axis3, long pulse1, long pulse2, long pulse3,  
long lspd, long hspd, double tacc);  
    int Sym_AbsoluteLine3(int axis1, int axis2, int axis3, long pulse1, long pulse2, long pulse3,  
long lspd, long hspd, double tacc);  
    int Sym_AbsoluteLine4(long pulse1, long pulse2, long pulse3, long pulse4, long lspd, long  
hspd, double tacc);
```

```
int Sym_RelativeLine4(long pulse1, long pulse2, long pulse3, long pulse4, long lspd, long
hspd, double tacc);
int Get_OutNum(int number);
int Manu_Pmove(int axis, long pulse);
int Manu_Continue(int axis);
int Manu_Disable(int axis);
int Setup_LockPosition(int axis, int mode, int regi, int logical);
int Get_LockStatus(int axis, int &v);
int Get_LockPosition(int axis, long &pos);
int Clr_LockPosition(int axis);
CCtrlCard();
int Result;          //return value
}
;# endif
```

(5) Source codes of the source file CtrlCard.cpp are as follows:

```
# include "stdafx.h"
# include "ADT8940A1.h"
# include "CtrlCard.h"
# include "VCExample.h"
extern int g_CardVer;
CCtrlCard::CCtrlCard()
{
}

/***** Initialization function *****/
'This function contain those library functions frequently used in control card
initialization, which is the foundation to call other functions and must be firstly called
in this example program.
'Return <=0 means initialization failure and >0 means initialization success
*****/

int CCtrlCard::Init_Board()
{
    Result = adt8940a1_initial();          //intial motion-card
    if (Result <= 0) return Result;
    for (int i = 1; i<=MAXAXIS; i++)
    {
```

//set limit mode, positive limit and negative limit is effective, low level is effective

```
set_limit_mode (0, i, 0, 0, 0);
```

```
set_command_pos (0, i, 0);           //set logic pos as 0
```

```
set_actual_pos (0, i, 0);           //set real pos as 0
```

```
set_startv (0, i, 1000);           //set start-speed
```

```
set_speed (0, i, 1000);           //set motion-speed
```

```
set_acc(0, i, 625);           //set acceleration
```

```
}
```

```
return 1;
```

```
}
```

```
/******set speed*****
```

according as para, judge whether is constant-speed

set start-speed, motion-speed and acceleration

para: axis -axis number

startv -start-speed

speed -motion-speed

add -acceleration

Return=0 correct, Return=1 wrong

```
*****/
```

```
int CCtrlCard::Setup_Speed(int axis, long startv, long speed, long add )
```

```
{
```

```
if (startv - speed >= 0) //constant-speed motion
```

```
{
```

```
Result = set_startv(0, axis, startv);
```

```
set_speed (0, axis, startv);
```

```
}
```

```
else                                //Trapezoidal acceleration/ deceleration
```

```
{
```

```
Result = set_startv(0, axis, startv);
```

```
set_speed (0, axis, speed);
```

```
set_acc (0, axis, add/125);
```

```
}
```

```
return Result;
```

```
}
```

```
/****** Single-axis motion function*****
```

This function is used to drive movement of a single axis

Return =0 means success, and Return =1 means error

*****/

```
int CCtrlCard::Axis_Pmove(int axis, long value)
```

```
{
```

```
    Result = pmove(0, axis, value);
```

```
    return Result;
```

```
}
```

/***** Any 2-axis interpolation function*****/

This function is used to drive any 2 axis to carry on linear interpolation

Return =0 means success, and Return =1 means error

*****/

```
int CCtrlCard::Inter_Move2(int axis1, int axis2, long value1, long value2)
```

```
{
```

```
    Result = inp_move2(0, axis1, axis2, value1, value2);
```

```
    return Result;
```

```
}
```

/***** Any 3-axis interpolation function*****/

This function is used to drive any 3 axis to carry on linear interpolation

Return =0 means success, and Return =1 means error

*****/

```
int CCtrlCard::Inter_Move3(int axis1, int axis2, int axis3, long value1, long value2, long value3)
```

```
{
```

```
    Result = inp_move3(0, axis1, axis2, axis3, value1, value2, value3);
```

```
    return Result;
```

```
}
```

/***** 4-axis interpolation function*****/

This function is used to drive the 4 axis to carry on linear interpolation

Return =0 means success, and Return =1 means error

*****/

```
int CCtrlCard::Interp_Move4(long value1, long value2, long value3, long value4)
```

```
{
```

```
    Result = inp_move4(0, value1, value2, value3, value4);
```

```
    return Result;
```

```
}
```

/***** Get information of movement *****/

This function is used to feedback the current logic position, real position and motion speed of the selected axis

Return =0 means success, and Return =1 means error

*****/

```
int CCtrlCard::Get_CurrentInf(int axis, long &LogPos, long &ActPos, long &Speed)
```

```
{
```

```
    Result = get_command_pos(0, axis, &LogPos);
```

```
    get_actual_pos(0, axis, &ActPos);
```

```
    get_speed(0, axis, &Speed);
```

```
    return Result;
```

```
}
```

/***** Stop motion function *****/

This function provides either sudden stop mode or deceleration stop mode

Return =0 means success, and Return =1 means error

*****/

```
int CCtrlCard::StopRun(int axis, int mode)
```

```
{
```

```
    if (mode == 0)
```

```
        Result = sudden_stop(0, axis);           //Sudden stop
```

```
    else
```

```
        Result = dec_stop(0, axis);              //Deceleration stop
```

```
    return Result;
```

```
}
```

```
/******Get motion status*****
```

This function is used to get single-axis drive status or interpolation drive status

Return =0 means success, and Return =1 means error

```
*****/
```

```
int CCtrlCard::Get_Status(int axis, int &value, int mode)
```

```
{
```

```
    if (mode==0)           //Get single-axis motion status
```

```
        Result=get_status(0,axis,&value);
```

```
    Else                   //Get motion status of interpolation
```

```
        Result=get_inp_status(0,&value);
```

```
    return Result;
```

```
}
```

2.3 Function realization module

2.3.1 Interface design

ADT-8940A1 DEMO
✖

ADT-8940A1 DEMO

Library Ver:110.0
Hardware Ver: 1.1

axis	start speed	run speed	add	add time
X	1000	2000	1500	0.1
Y	1000	2000	1500	0.1
Z	1000	2000	1500	0.1
A	1000	2000	1500	0.1

delay
 delay time: us
 ☐ delay status

external drive
☐ Manu_Pmove

axis	pulse	logic pos	actual pos	run speed	LMT-	LMT+	STOP0	STOP1
☐ X	100000	0	0	0	☐	☐	☐	☐
☐ Y	100000	0	0	0	☐	☐	☐	☐
☐ Z	100000	0	0	0	☐	☐	☐	☐
☐ A	100000	0	0	0	☐	☐	☐	☐

lock pos

Remark:

- (1) Speed setting part—used to set starting speed, drive speed and acceleration of every axis; position setting—used to set drive pulse for every axis; motion information—used to real-time display logical position, real position and motion speed of every axis;
- (2) Drive object—users determine axis joining simultaneous movement or interpolation by selecting drive objects;
- (3) Simultaneous movement—Used to send single-axis drive commands to all the axis of the selected drive object; interpolation —Used to send interpolation command to all the axis of the selected drive object; stop—stop all the pulse outputs of all axis.

All the above data take pulse as the unit.

2.3.2 Initialization codes for the movement control card are inside window initialization, while users shall supplement the following codes:

```

int i=g_CtrlCard.Init_Board();
//*****initial 8940A1 motion-card*****
if (i <= 0)
{

```

```
        MessageBox( "Fail to initial motion-card!");
        if (i==0)
        {
            MessageBox( "NO installation of ADT8940A1!");
        }
        if(i== -1)
        {
            MessageBox( "no installation of service!");
        }
        if(i== -2)
        {
            MessageBox( "PCI bus failure!");
        }
    }
else
    MessageBox ("Succeed in initial motion-card!");

//*****Get Version*****
float LibVer;           //Library Version
float HardwareVer;      //Hardware Version
g_CtrlCard.Get_Version(LibVer,HardwareVer);
CStatic *lbl;
CString str;
lbl=(CStatic*)GetDlgItem(IDC_INFO_VER);
str.Format("Library Ver:%1.1f\nHardware Ver:
%1.1f",LibVer,HardwareVer);
lbl->SetWindowText(str);

//***** set start-speed 1000 *****
m_nStartvX = 1000;
m_nStartvY = 1000;
m_nStartvZ = 1000;
m_nStartvA = 1000;
//*****set run-speed 2000*****
m_nSpeedX  = 2000;
m_nSpeedY  = 2000;
m_nSpeedZ  = 2000;
m_nSpeedA  = 2000;
//*****set add 1500*****
m_nAddX    = 1500;
```

```
m_nAddY    = 1500;
m_nAddZ    = 1500;
m_nAddA    = 1500;
//*****set pulse 100000*****
m_nPulseX  = 100000;
m_nPulseY  = 100000;
m_nPulseZ  = 100000;
m_nPulseA  = 100000;
//*****set add time*****
m_dTaccX   = 0.1;
m_dTaccY   = 0.1;
m_dTaccZ   = 0.1;
m_dTaccA   = 0.1;
//*****set delay time 0*****
m_nDelayTime = 0;
UpdateData(FALSE);
//*****set time*****
SetTimer(MAINTIMER,100,NULL);
```

2.3.3 Simultaneous movement codes are inside the click message of Simultaneous movement button and will send various drive commands for various selected targets; the codes are as follows:

```
void CDEMODlg::OnButtonPmove()
```

```
{
    UpdateData(TRUE);
    Long Startv[]={m_nStartvX,m_nStartvY,m_nStartvZ,m_nStartvA}; //start-speed
    long Speed[]={m_nSpeedX,m_nSpeedY,m_nSpeedZ,m_nSpeedA};      //run-speed
    long Add[]  = {m_nAddX,m_nAddY,m_nAddZ,m_nAddA};              //add
    if(m_bX)
    {
        //*****set speed*****//
        g_CtrlCard.Setup_Speed(1, m_nStartvX, m_nSpeedX, m_nAddX);
        //*****X axis move*****//
        g_CtrlCard.Axis_Pmove(1, m_nPulseX);
    }
}
```

```
if(m_bY )
{
    //*****set speed*****//
    g_CtrlCard.Setup_Speed(2, m_nStartvY, m_nSpeedY, m_nAddY);
    //*****Y axis move*****//
    g_CtrlCard.Axis_Pmove(2, m_nPulseY);
}
if(m_bZ )
{
    //*****set speed*****//
    g_CtrlCard.Setup_Speed(3, m_nStartvZ, m_nSpeedZ, m_nAddZ);
    //*****Z axis move*****//
    g_CtrlCard.Axis_Pmove(3, m_nPulseZ);
}
if(m_bA )
{
    //*****set speed*****//
    g_CtrlCard.Setup_Speed(4, m_nStartvA, m_nSpeedA, m_nAddA);
    //*****A axis move*****//
    g_CtrlCard.Axis_Pmove(4, m_nPulseA);
}
}
```

2.3.4 Interpolation codes are inside the click message of the inp_move button and will send various drive commands for various selected targets; the codes are as follows:

```
void CDEMOMDlg::OnButtonInpmove ()
```

```
{
    UpdateData(TRUE);
    long startv[]={m_nStartX,m_nStartY,m_nStartZ,m_nStartW};
    long Speed[]={m_nXSpeed,m_nYSpeed,m_nZSpeed,m_nWSpeed};
    long Add[]={m_nAddX,m_nAddY,m_nAddZ,m_nAddW};
}
```

```
long Pos[]={m_nPosX,m_nPosY,m_nPosZ,m_nPosW};  
if(m_bX && m_bY && m_bZ && m_bW){           //Interpolation of the 4 axis  
    g_CtrlCard.Setup_Speed(1,startv[0],Speed[0],Add[0]);  
    g_CtrlCard.Interp_Move4(Pos[0],Pos[1],Pos[2],Pos[3]);  
}  
else if(m_bX && m_bY && m_bZ){                //XYZ Interpolation of 3 axis  
    g_CtrlCard.Setup_Speed(1,startv[0],Speed[0],Add[0]);  
    g_CtrlCard.Interp_Move3(1,2,3,Pos[0],Pos[1],Pos[2]);  
}  
else if(m_bX && m_bY && m_bW){                //XYW Interpolation of 3 axis  
    g_CtrlCard.Setup_Speed(1,startv[0],Speed[0],Add[0]);  
    g_CtrlCard.Interp_Move3(1,2,4,Pos[0],Pos[1],Pos[3]);  
}  
else if(m_bX && m_bZ && m_bW){                //XZW Interpolation of 3 axis  
    g_CtrlCard.Setup_Speed(1,startv[0],Speed[0],Add[0]);  
    g_CtrlCard.Interp_Move3(1,3,4,Pos[0],Pos[2],Pos[3]);  
}  
else if(m_bY && m_bZ && m_bW){                //YZW Interpolation of 3 axis  
    g_CtrlCard.Setup_Speed(2,startv[1],Speed[1],Add[1]);  
    g_CtrlCard.Interp_Move3(2,3,4,Pos[1],Pos[2],Pos[3]);  
}  
else if(m_bX && m_bY){                        //XY Interpolation of 2 axis  
    g_CtrlCard.Setup_Speed(1,startv[0],Speed[0],Add[0]);  
    g_CtrlCard.Interp_Move2(1,2,Pos[0],Pos[1]);  
}  
else if(m_bX && m_bZ){                        //XZ Interpolation of the 2 axis  
    g_CtrlCard.Setup_Speed(1,startv[0],Speed[0],Add[0]);  
    g_CtrlCard.Interp_Move2(1,3,Pos[0],Pos[2]);  
}  
else if(m_bX && m_bW){                        //XW Interpolation of the 2 axis  
    g_CtrlCard.Setup_Speed(1,startv[0],Speed[0],Add[0]);
```

```
        g_CtrlCard.Interp_Move2(1,4,Pos[0],Pos[3]);
    }
    else if(m_bY && m_bZ){    //YZ Interpolation of the 2 axis
        g_CtrlCard.Setup_Speed(2,startv[1],Speed[1],Add[1]);
        g_CtrlCard.Interp_Move2(2,3,Pos[1],Pos[2]);
    }
    else if(m_bY && m_bW){    //YW Interpolation of the 2 axis
        g_CtrlCard.Setup_Speed(2,startv[1],Speed[1],Add[1]);
        g_CtrlCard.Interp_Move2(2,4,Pos[1],Pos[3]);
    }
    else if(m_bZ && m_bW){    //ZW Interpolation of the 2 axis
        g_CtrlCard.Setup_Speed(3,startv[2],Speed[2],Add[2]);
        g_CtrlCard.Interp_Move2(3,4,Pos[2],Pos[3]);
    }
}
```

2.4 Monitoring module

The monitoring module is used to real-time get drive information of all the axes and display movement information, at the same time of controlling them in drive process without any new drive commands. This module is completed through timer messages, with the following codes:

```
void CDEMOMDlg::OnTimer(UINT nIDEvent)
{
    long log=0,act=0,spd=0;
    UINT
    nID1[]={IDC_POS_LOGX,IDC_POS_LOGY,IDC_POS_LOGZ,IDC_POS_LOGA};
    UINT
    nID2[]={IDC_POS_ACTX,IDC_POS_ACTY,IDC_POS_ACTZ,IDC_POS_ACTA};
    UINT
    nID3[]={IDC_RUNSPEED_X,IDC_RUNSPEED_Y,IDC_RUNSPEED_Z,IDC_RUNSPEED_A};
};

CStatic *lbl;
CString str;
```

```

int status[4];
for (int i=1; i<MAXAXIS+1; i++)
{
    g_CtrlCard.Get_CurrentInf(i,log,act,spd);    //Get logic-pos ,actual-pos
and run-speed
    //*****display logic-pos*****//
    lbl=(CStatic*)GetDlgItem(nID1[i-1]);
    str.Format("%ld",log);
    lbl->SetWindowText(str);
    //*****display actual-pos*****//
    lbl=(CStatic*)GetDlgItem(nID2[i-1]);
    str.Format("%ld",act);
    lbl->SetWindowText(str);
    //*****display run-speed*****//
    lbl=(CStatic*)GetDlgItem(nID3[i-1]);
    str.Format("%ld",spd);
    lbl->SetWindowText(str);
    //*****Get status*****//
    g_CtrlCard.Get_Status(i,status[i-1],0);
}
//*****Check signal*****//
//  XLMT    -0    XLMT+   -1
//  XSTOP0 -2    XSTOP1  -3
//  YLMT    -6    YLMT+   -7
//  YSTOP0 -8    YSTOP1  -9
//  ZLMT    -12   ZLMT+   -13
//  ZSTOP0 -14   ZLMT+   -15
//  ALMT    -18   ALMT+   -19
//  ASTOP0 -20   ASTOP1  -21
//*****
UINT nIDIN[]={ IDC_LIMIT_X,IDC_LIMIT_X2,           //XLMT+/XLMT-
                IDC_STOP0_X, IDC_STOP1_X ,
                IDC_LIMIT_Y, IDC_LIMIT_Y2,          //YLMT+/YLMT-
                IDC_STOP0_Y,IDC_STOP1_X2,
                IDC_LIMIT_Z,IDC_LIMIT_Z2,           //ZLMT+/ZLMT-
                IDC_STOP0_Z,IDC_STOP1_Z,
                IDC_LIMIT_A,IDC_LIMIT_A2,           //ALMT+/ALMT-
                IDC_STOP0_A,IDC_STOP1_A,
                };
int io[]={0,1,2,3,6,7,8,9,12,13,14,15,18,19,20,21};

```

```
    CButton *btn;
    int value;
    for (i=0; i<16; i++)
    {
        value=g_CtrlCard.Read_Input(io[i]);           //read input
signal
        btn=(CButton*)GetDlgItem(nIDIN[i]);
        btn->SetCheck(value==0?1:0);
    }
    //*****contorl
button*****
    if(status[0]==0 && status[1]==0 && status[2]==0 && status[3]==0)
    {
        //*****finished*****
        btn=(CButton*)GetDlgItem(IDC_BUTTON_PMOVE);
        btn->EnableWindow(TRUE);
        btn=(CButton*)GetDlgItem(IDC_BUTTON_INPMOVE);
        btn->EnableWindow(TRUE);
        btn=(CButton*)GetDlgItem(IDC_BUTTON_CLEARPOS);
        btn->EnableWindow(TRUE);
        btn=(CButton*)GetDlgItem(IDC_BUTTON_BASEPARA);
        btn->EnableWindow(TRUE);
        btn=(CButton*)GetDlgItem(IDC_BUTTON_IOTEST);
        btn->EnableWindow(TRUE);
        btn=(CButton*)GetDlgItem(IDC_LINEINP_MOVE);
        btn->EnableWindow(TRUE);
        btn=(CButton*)GetDlgItem(IDC_COMP_MOVE);
        btn->EnableWindow(TRUE);
    }
    else
    {
        //*****running*****
        btn=(CButton*)GetDlgItem(IDC_BUTTON_PMOVE);
        btn->EnableWindow(FALSE);
        btn=(CButton*)GetDlgItem(IDC_BUTTON_INPMOVE);
        btn->EnableWindow(FALSE);
        btn=(CButton*)GetDlgItem(IDC_BUTTON_CLEARPOS);
        btn->EnableWindow(FALSE);
        btn=(CButton*)GetDlgItem(IDC_BUTTON_BASEPARA);
        btn->EnableWindow(FALSE);
    }
}
```



```
        btn=(CButton*)GetDlgItem(IDC_BUTTON_IOTEST);  
        btn->EnableWindow(FALSE);  
        btn=(CButton*)GetDlgItem(IDC_LINEINP_MOVE);  
        btn->EnableWindow(FALSE);  
        btn=(CButton*)GetDlgItem(IDC_COMP_MOVE);  
        btn->EnableWindow(FALSE);  
    }  
    CDialog::OnTimer(nIDEvent);  
}
```

2.5 Stop module

This module is mainly used to control unexpected events during drive process and will immediately stop drive of all the axes. Codes of this stop module are within the click messages of Stop button, with the following codes:

```
void CDEMOMlg::OnButtonStoprun()  
{  
    for (int i = 1; i<= MAXAXIS; i++){  
        g_CtrlCard.StopRun(i,0);  
    }  
}
```

Chapter 11 Normal failures and solutions

Movement control card detection failure

During use of control card, if encountering failure to detect the control card, users may follow the following items to check:

- (1) Check whether drive program for the control card has been installed step by step following installation guide and whether there is the dynamic library file for the control card under the system menu (System32 or System);
- (2) Check touch between the movement control card and the slot; users may test it by re-inserting or changing the slot, alternatively, use a rubber to clean dirt on the golden finger of the control card and re-insert;
- (3) Under the system equipment manager, check whether there is conflict between the movement control card and other hardware. In case of use of PCI card, users may remove other cards or boards first, such as sound card and network card; in

case of PC104 card, users may adjust the dialing switch and reset the base address, while the base address used during card initialization must be same as the actual base address;

- (4) Check whether there are any problems with the operating system; users may test it through re-installing other versions of operating systems;
- (5) If failing to find the control card after the above steps, users may change the control card for further detection so as to discover whether there is damage with the control card.

MOTOR SERVICE FAILURE

In case the motor breakdowns while the movement control card works normally, users may follow the following points for troubleshooting.

- (1) Motor makes no reaction when the movement control card outputs pulses
 - Check cable between the control card and the terminal panel;
 - Check whether the pulse and direction signal wire of the motor driver has been correctly connected to the terminal panel;
 - Check connection of the external power supply for the servo driver;
 - Check whether there is alarming status in the servo/ stepping motor driver; in case of any alarm there, follow codes corresponding to alarms to check the reason.
 - Check connection to the servo SON and whether there is excitation status in the servo motor ;
 - In case of servo motor, check control method of the driver; control card of our company support the Position Control Method.
 - Damage to the motor/ driver
- (2) Stepping motor makes abnormal noise during service and motor makes obvious out-steps.
 - Calculate motor speed and make sure the stepping motor is under 10-15 rounds per second instead of faster speed;
 - Check internal obstruction in the mechanical part or resistance to the machine;

- Change to large-moment motors if the current motor is not sufficient;
- Check current and voltage of the driver; current shall be set as 1.2 of the nominated current and supply voltage shall be within the nominated range;
- Check the starting speed of the controller; normal starting speed shall be 0.5-1 and the acceleration/ deceleration time shall be over 0.1 second.

(3) Servo/ stepping motor makes obvious vibration or noises during processing

- Reduce the position ring gain and speed ring gain of the driver while allowed by the positioning precision, if the cause is such ring gains are too big;
- Adjust machine structure if the cause is poor machine rigidity;
- Change to large-moment motors if the current motor is not sufficient;
- Avoid the co-vibration area of the motor or increase partitions so as not to have the speed of stepping motor within the co-vibration area of the motor.

(4) Motor positions inaccurately

- Check whether the mechanic screw pitch and pulses per round comply with the parameters set in the actual application system, i.e., pulse equivalent;
- Enlarge position ring gain and speed ring gain in case of servo motor;
- Check screw gap of the machine in the way of measuring the backward gap of a screw through a micrometer and adjust the screw if there is any gap;
- In case of inaccurate positioning out of regular time or position, check external disturbance signals;
- Check whether it is due to non-powerful motor that there is shaking or out-step.

(5) Motor makes no direction

- Check DR+ DR- cable for connection error or loose connection;
- Make sure the pulse mode applied in the control card comply with the actual driver mode; this control card support either “pulse + direction” or “pulse + pulse” mode.
- Check broken cable or loose connection along the motor cable, in case of stepping motor.

ABNORMAL SWITCH AMOUNT INPUT

In case some input signals give unusual detection results during system adjusting and running, users may check in accordance with the following methods:

(1) No signal input

- Check whether the wiring is correct according to the above-introduced wiring maps for normal switch and approach switch and ensure the public port for photoelectric coupling of input signals have been connected with anode of internal or external power supply (+12V or 24V);
- Check switch model and wiring method; the input switch for I/O points of our company is of NPN model.
- Check whether there is damage with the photoelectric coupler. In case of normal wiring, input status will not change no matter the input point is broken or closed; users may use multi-use meter to check whether the photoelectric coupler has been broken, and if yes, replace with a new one;
- Check the 12V or 24V power supply to the switch;
- Check whether there is damage to the switch.

(2) Non-continuous signals

- Check whether there is disturbance by detecting signal status in the I/O test interface; in case of disturbance, increase with Model 104 multiple layer capacitor or apply blocking cables;
- If the machine makes obvious shaking or unusual work stop during normal service, check whether there is disturbance to the limit switch signals or the limit switch work reliably;
- Check connection of external cables.

(3) Inaccurate reset

- Too high speed decreases reset speed ;
- Check disturbance source if the problem is there is external disturbance to signals;
- Wrong resetting direction;
- Improper installation position of the reset switch or loose switch

(4) Limit out of use

- Check whether the limit switch still works under the I/O test;
- Too high speed during manual or automatic processing;
- Check disturbance source if the problem is there is external disturbance to signals;
- Wrong manual direction;
- Improper installation position of the reset switch or loose switch

 Abnormal output of switch amount

Abnormal output of switch amount may be checked in the following method:

(1) Abnormal output

- Check whether the wiring is correct following the above-introduced wiring for output points and ensure the output public port (earthing line) has been connected with the earthing line of the to-be-used power supply;
- Check whether there is any damage to the output components;
- Check whether there is damage with the photoelectric coupler. Users may use multi-use meter to check whether the photoelectric coupler has been broken, and if yes, replace with a new one;
- Safety issue. Continuous dioxide (model: IN4007 or IN4001) must be serial connected in case of output with sensitive loading.

(2) Judgment method for improper output

Break the external cable at the output point and connect at the output point a pull-up resistor of around 10K to the power supply, while earthing line of the output must be connected with GND of the power supply; then users use the red pen of a multi-use meter to touch the 12V anode, and black pen to touch the signal output port, at the same time of using hand to touch the button on the test interface to see whether there is voltage output; in case of any voltage output, check the external circuit, otherwise check connection to the public port of boards/ cards and internal photoelectric couplers.

Abnormal encoder performance

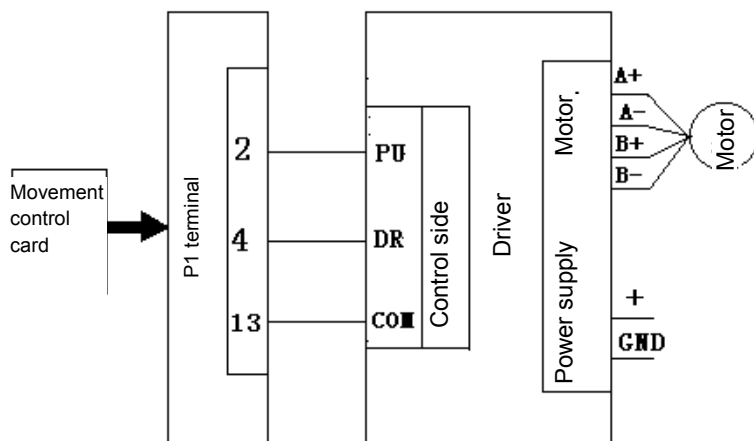
Abnormal encoder performance may be checked in the following method:

- (1) Check encoder cables and make sure they comply with the above-introduced differential or collecting electrode wiring method;
- (2) Check encoder voltage. The movement control card normally accepts +5V signals. In case a +12V or +24V encoder is selected, users must serial connect a 1K (+12V) resistor between the Phase A /B of the encoder and Phase A /B of the terminal panel;
- (3) Inaccurate encoder counting. External cables to the encoder must be blocking double-twisted cables, and shall be tied free from those cables with strong disturbance such as strong electricity, specifically, they shall be separated for over 30~50MM.

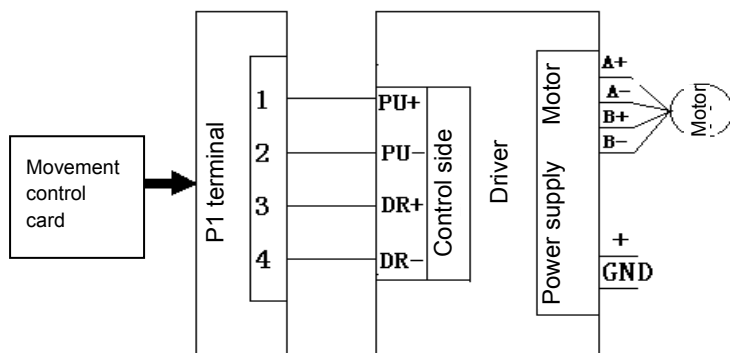
Appendix A Typical wiring for motor driver

All the following wiring takes X axis as example.

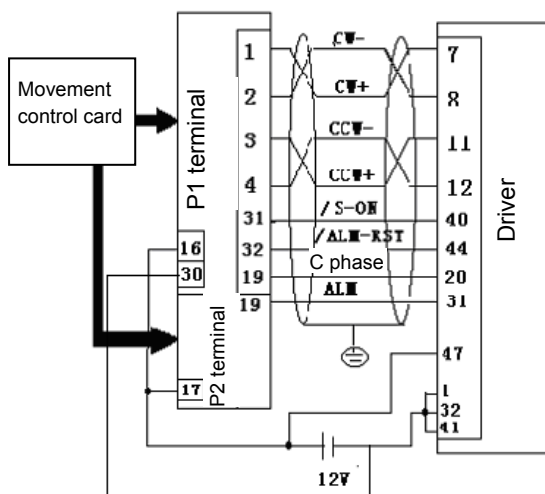
Stepping motor driver common anode wiring



Stepping motor driver differential wiring



Yaskawa servo driver wiring



Panasonic A4 servo driver wiring

