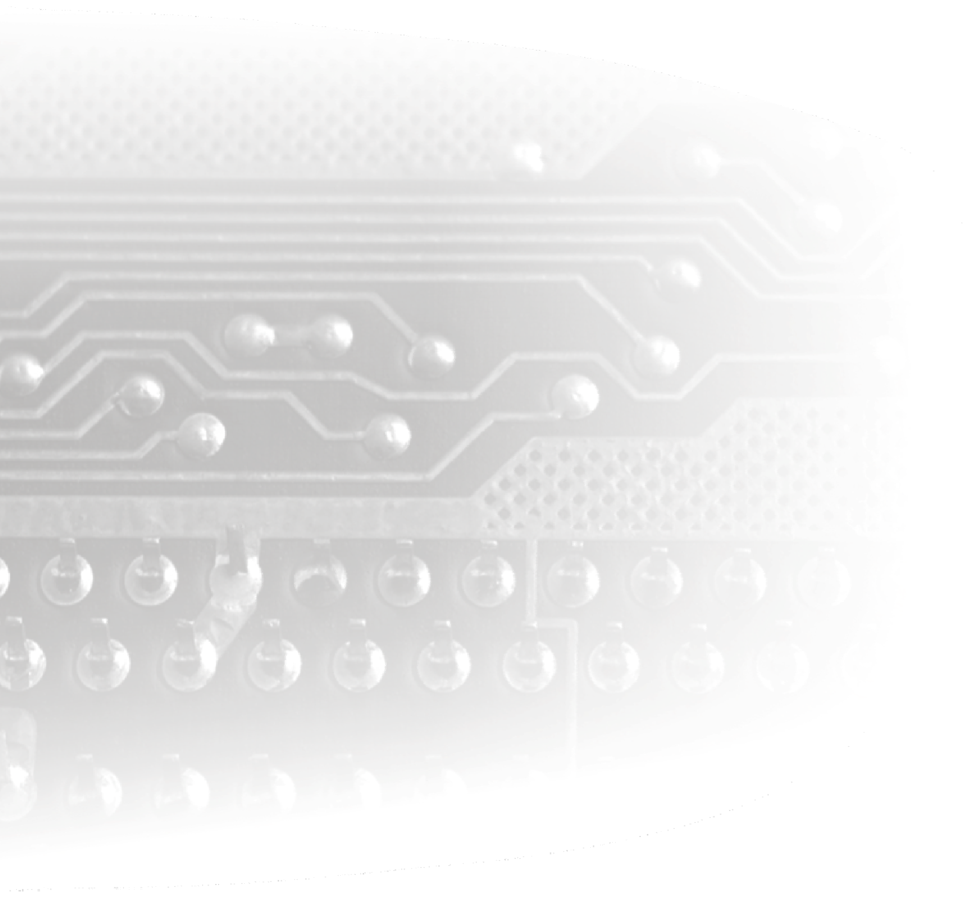
ADT-8860

PMC Programmable Motion Controller

User's Manual



ADTECH (SHENZHEN) CNC TECHNOLOGY CO., LTD

5th Floor, 27-29th Building, Tianxia IC Industrial Park, Yiyuan road, Nanshan District,

Shenzhen Post code: 518052

Tel:86-755-26722719 Fax: 86-755-26722718

E-mail: export@adtechen.com http://www.adtechen.com

Copyright

Version Update

Project No.	Version No.	Revision Date	Description
ADT-8860	A0101	2011-8-15	1st

Remark: The three digits in version number have the following meanings:

$\boxed{0}$ $\boxed{1}$ $\boxed{0}$

Main version number of library / Secondary version number of library / Reserved

Remark:

1. Adtech has collated and checked this Manual strictly; however, we can't guarantee that there is no error or omission in this Manual.

2. Adtech is committed to continuously improving product function and service quality. The product, software and Manual are subject to changes without prior notice.

Contents

1. Product Overview

ADT-8860 is a multipurpose and onsite programmable six axes motion controller (PMC -- program motion control) with PLC function. It supports offline operating mode, online real-time operating mode and console operating mode, and is widely used in various automation control equipment.

When the controller is running in offline mode, it used C similar language style instruction programming and control, and is partly compatible with G code motion instructions. When it is in online running mode, any third party software can access PMC internal system register area through standard Modbus communication protocol to control the various function of the motion control card and provide flexible open architecture.

1.1. Hardware system description

- This system uses SAMSUNG series 32-bit high performance processor Samsung S3C2440AL with 400MHz main frequency;
- SDRAM memory, onboard 64MB SDRAM (standard configuration), SDRAM clock frequency is up to 100MHz;
- Nand Flash onboard 256MB Nand Flash (standard configuration), nonvolatile, sufficient storage capacity for processing files, satisfying the running and processing requirement of complicated programs;
- CorePower professional 1.25V core voltage power supply, solving the CPU heating problem perfectly;
- Use super large scale programmable device FPGA and real-time multitask control technology and hardware interpolation technology to ensure high stability;
- Output pulse frequency is up to 2M, interpolation precision is 0.5 pulse;
- 86 lines input IO, 40 lines output IO, full optical coupling isolation;
- Support step motor and servo motor;
- Support USB1.1 interface, allow user updating new software to USB disk at any moment or copying latest application from USB disk to the controller, ensuring convenient on-site operation;
- Reasonable structure, all coupling isolation control, strong anti-interference capacity;
- 3 lines RS232 (±15KV ESD protection)
- Standard industrial CAN bus interface (reserved)
- Support handheld box input (reserved)
- Standard network interface
- Buzzer prompt

1.2. Product specifications

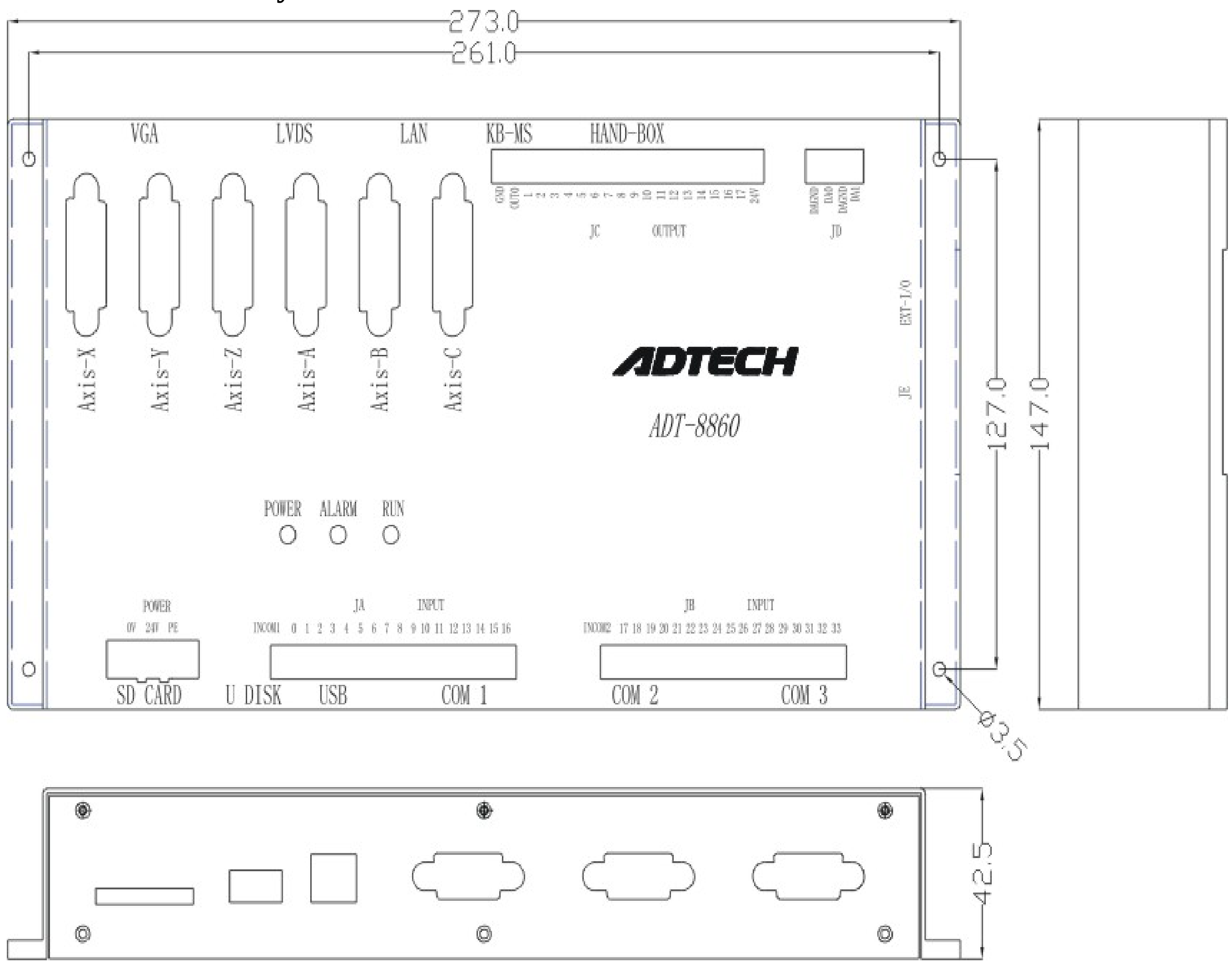
Function	Name	Description
Control axis	Number of control axes	6 axes (X, Y, Z, A, B, C)
	Number of simultaneous control axes	Random 1-6 axes linkage Random 1-3 axes linear interpolation Random 2-3 axes arc helical interpolation 4-6 fixed axes linear interpolation
Communication mode	Ethernet interface	TCP/IP Modbus communication
	R232 serial port 1	Command console
	R232 serial port 2	R232 Modbus communication
	USB interface	For system update and file transmission
Motion characteristics	Minimum setting unit	0.001mm
	Minimum motion unit	0.001mm
	Maximum instruction value	±9999.999mm
	Feed speed range	1~9999mm/min
	Linear acceleration/deceleration	Yes
	S acceleration/deceleration	Enable at look-ahead algorithm
	Look-ahead	Yes
	Return to reference point	Return to reference point synchronously or step by step
Motion instruction	Positioning, linear interpolation, arc interpolation	G00,G01,G02/G03 MOVE,LINE,ARC_A,ARC_B
Operating mode	Offline operating	Execute up to 32 programs synchronously
	Online control	Create work network with up to 64 controllers through Ethernet
	Console human - computer interaction	Run in command line input mode
Coordinate	Pause and resume motion	Yes

system and pause	Instruction delay function	≥1ms
	Coordinate system setting	Support 32 workpiece coordinate systems
	Automatic coordinate system setting	Yes
Safety function	Soft/hard limit check	Yes
	Emergency stop	Yes
Program storage	Program storage capacity, number of stored program	Total capacity: 256MB, single file up to 1MB, no limit on quantity of processing files
Program control	Sequence, loop, goto, subroutine call	Yes
	Decimal programming	Yes
Display	None	Use PC or third party human – computer interface
Axis configuration	Configuration of physical axis and logic axis conversion	Yes

1.3. Operating environment

Operating voltage	24V DC (with filter)
Operating temperature	0℃— 45℃
Optimum operating temperature	5℃— 40℃
Operating humidity	10%——90%, no condensation
Optimum operating humidity	20%——85%
Storage temperature	0℃—50℃
Storage humidity	10%——90%

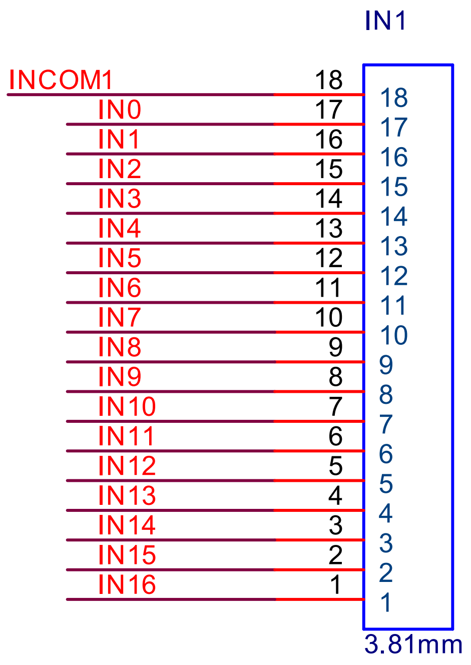
1.4. Installation layout



2. Electrical Interface and Connection

2.1. IN1 wire No. description

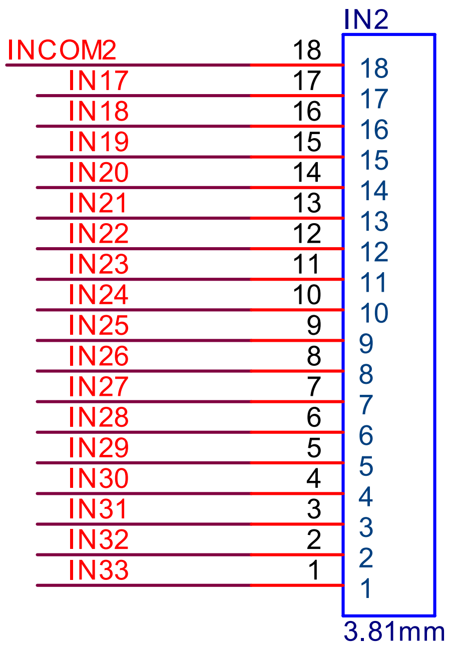
(17 lines input; refer to the silkscreen on the board for the sequence)



Wire No.	Name	Function
1	IN16	Universal input IO
2	IN15	Universal input IO
3	IN14	Universal input IO
4	IN13	Universal input IO
5	IN12	Universal input IO
6	IN11	A negative limit /universal input IO
7	IN10	A positive limit / universal input IO
8	IN9	Z negative limit / universal input IO
9	IN8	Z positive limit / universal input IO
10	IN7	Y negative limit / universal input IO
11	IN6	Y positive limit / universal input IO
12	IN5	X negative limit / universal input IO
13	IN4	X positive limit / universal input IO
14	IN3	A home / universal input IO
15	IN2	Z home / universal input IO
16	IN1	Y home / universal input IO
17	IN0	X home / universal input IO
18	INCOM1	17 lines input point common terminal 1

2.2. IN2 wire No. description

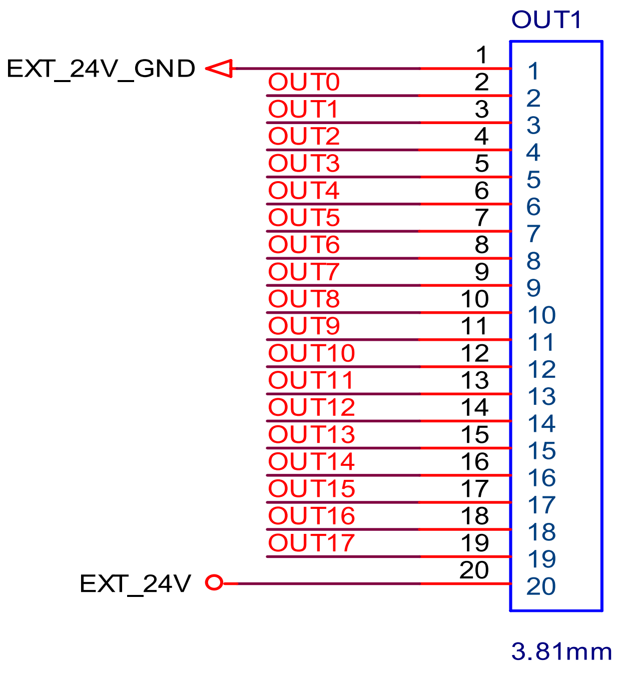
(17 lines input; refer to the silkscreen on the board for the sequence)



Wire No.	Name	Function
1	IN33	Universal input IO (refer to the silkscreen on the board for the sequence)
2	IN32	Universal input IO
3	IN31	Universal input IO
4	IN30	Universal input IO
5	IN29	Universal input IO
6	IN28	Universal input IO
7	IN27	Universal input IO
8	IN26	Universal input IO
9	IN25	Universal input IO
10	IN24	Universal input IO
11	IN23	Universal input IO
12	IN22	C negative limit / universal input IO
13	IN21	C positive limit / universal input IO
14	IN20	B negative limit / universal input IO
15	IN19	B positive limit / universal input IO
16	IN18	C home / universal input IO
17	IN17	B home / universal input IO
18	INCOM2	Input common terminal 2 (please connect it to +12V~ +24V power supply)

2.3. OUT1 wire No. description

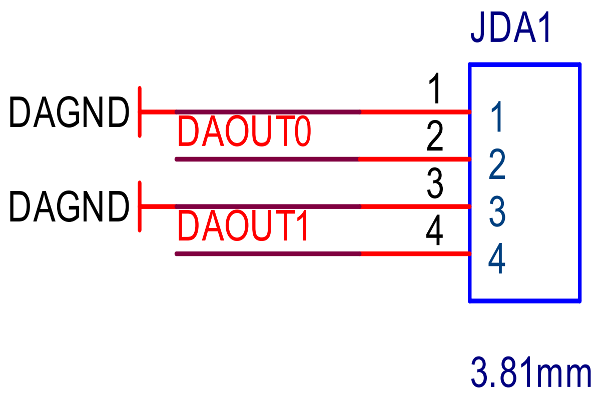
(18 lines output; refer to the silkscreen on the board for the sequence)



Wire No.	Name	Function
1	24VGND	Output common terminal
2	OUT0	Output 0--17
3	OUT1	
4	OUT2	
5	OUT3	
6	OUT4	
7	OUT5	
8	OUT6	
9	OUT7	
10	OUT8	
11	OUT9	
12	OUT10	
13	OUT11	
14	OUT12	
15	OUT13	
16	OUT14	
17	OUT15	
18	OUT16	
19	OUT17	
20	+24V	Load +24 power input terminal (require external +12~ +24V power supply)

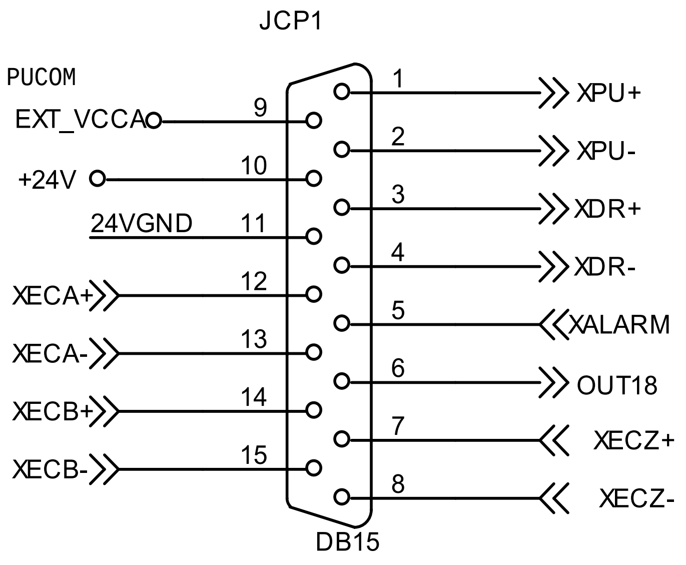
2.4. JDA1 wire No. description

(Two lines analog voltage output; refer to the silkscreen on the board for the sequence)



1	DAGND	Analog voltage reference ground
2	DAOUT0	Analog voltage line 0
3	DAGND	Analog voltage reference ground
4	DAOUT1	Analog voltage line 1

2.5. X1, Y1, Z1, A1, B1, C1 drive control interface wire No. description

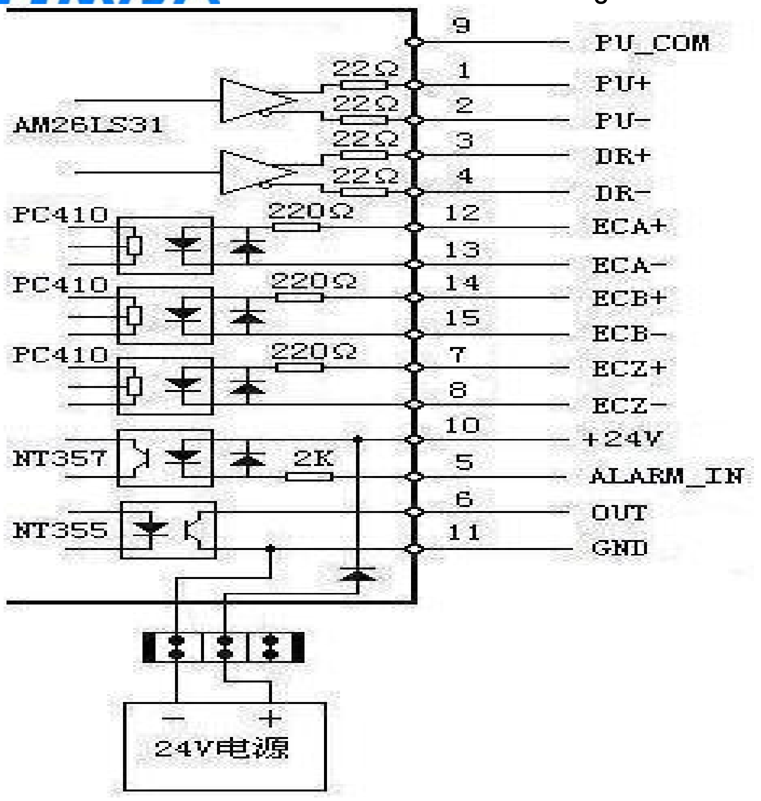


15脚母头（与车铣床相同）

15-pin female connector (same to milling machine)

Wire No.	Definition	Function
1	nPU+	Pulse signal +
2	nPU-	Pulse signal -
3	nDR+	Direction signal +

4	nDR-	Direction signal -
5	nALARM	Universal input point, used for alarm input (X-34 Y-35 Z-36 A-37 B-38 C-39)
6	OUTn	Universal output point (X-18 Y-19 Z-20 A-21 B-22 C-23)
7	nECZ+	Encoder Z phase input + (X-42 Y-45 Z-48 A-51 B-54 C-57)
8	nECZ-	Encoder Z phase input -
9	PUCOM	Used for drive of single terminal input
10	EXT_24V	Provide 24V power supply externally
11	EXT_24V_GND	
12	nECA+	Encoder A phase input + (X-40 Y-43 Z-46 A-49 B-52 C-55)
13	nECA-	Encoder A phase input -
14	nECB+	Encoder B phase input + (X-41 Y-44 Z-47 A-50 B-53 C-56)
15	nECB-	Encoder B phase input -

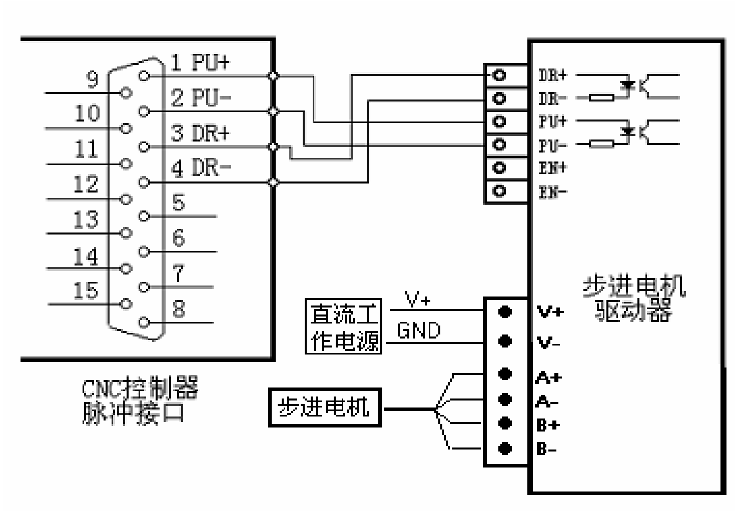


24V 电源 24V power supply

Brief Internal Circuit Diagram of Pulse Output

➢ Wiring mode of step motor drive with differential input

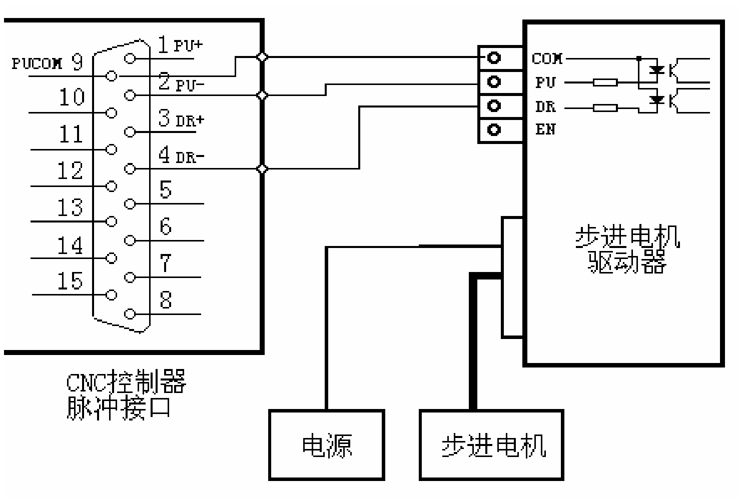
Take Adtech drive for example. All Adtech drives are in differential input mode, which is recommended due to its strong anti-interference. The wiring of CNC, step motor drive and step motor follows



CNC 控制器脉冲接口 CNC controller pulse interface 直流工作电源 DC power supply
步进电机驱动器 Step motor drive 步进电机 Step motor

➢ Wiring diagram of step motor drive with single input

The step drives of certain company connect the cathode of optical coupling input, i.e. common cathode connection, which isn’t suitable for this controller. It is common anode connection to connect together the anodes of optical coupling. In this case, please connect the wires according to the figure below, and do not connect PU+ and DR+ together, or else pulse interface will be damaged.



CNC 控制器脉冲接口 CNC controller pulse interface 步进电机驱动器 Step motor drive
电源 Power supply 步进电机 Step motor

Wiring diagram of step motor drive with common anode input

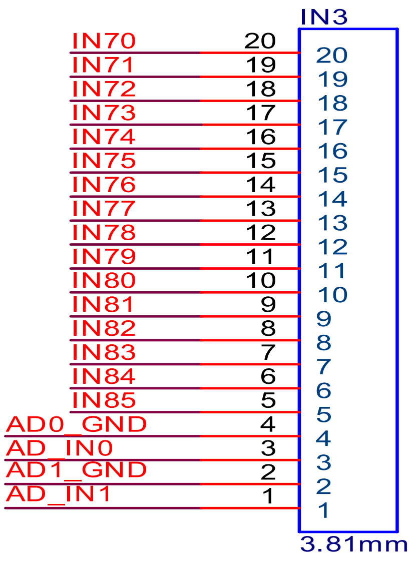
➢ **Wiring diagram with servo motor drive**

Since most are in differential connection, please refer to the connection of differential mode for the pulse part. Many servo drives require 12-24V power supplies and can use the 24V power supply provided by pin 10&11. The specific connection depends on the servo drive. Please contact us if there any question.

Note: Either two of PU+, PU-, DR+ and DR- can't be connected directly, or else the internal circuit may be damaged.

2.6. IN3 wire No. description

(16 lines input; refer to the silkscreen on the board for the sequence)

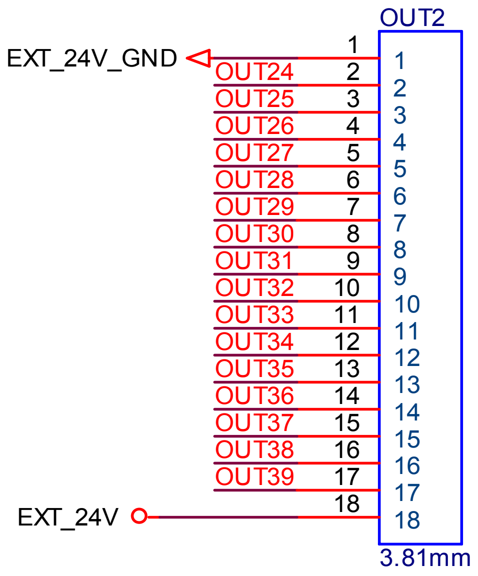


Wire No.	Name	Function

1	AD_IN1	Analog input line 2
2	AD1_GND	Reference ground of analog input line 2
3	AD_IN0	Analog input line 1
4	AD0_GND	Reference ground of analog input line 1
5	IN85	Universal input IO
6	IN84	
7	IN83	
8	IN82	
9	IN81	
10	IN80	
11	IN79	
12	IN78	
13	IN77	
14	IN76	
15	IN75	
16	IN74	
17	IN73	
18	IN72	
19	IN71	
20	IN70	

2.7. OUT2 wire No. description

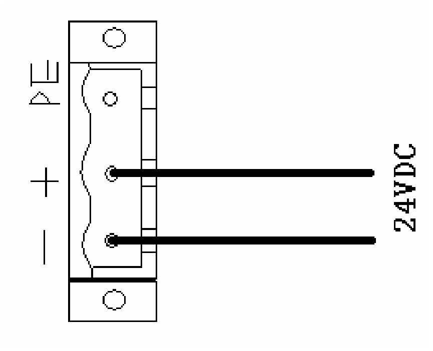
(16 lines output; refer to the silkscreen on the board for the sequence)



Wire No.	Name	Function
1	EXT_24V_GND	Power ground
2	OUT24	Universal output 0--17
3	OUT25	
4	OUT26	
5	OUT27	
6	OUT28	
7	OUT29	
8	OUT30	
9	OUT31	
10	OUT32	
11	OUT33	
12	OUT34	
13	OUT35	
14	OUT36	
15	OUT37	
16	OUT38	
17	OUT39	
18	EXT_24V	+24 power input terminal (require external switching power supply)

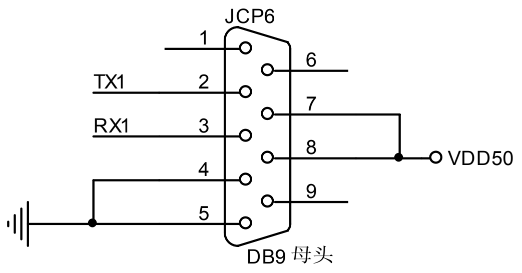
2.8. Power wiring mode

The power supply of the mainboard is +24V, which is provided by external switching power supply. The power wiring follows:



2.9. Serial port 1

Serial port 0 is usually used for system update, console testing and system info display.

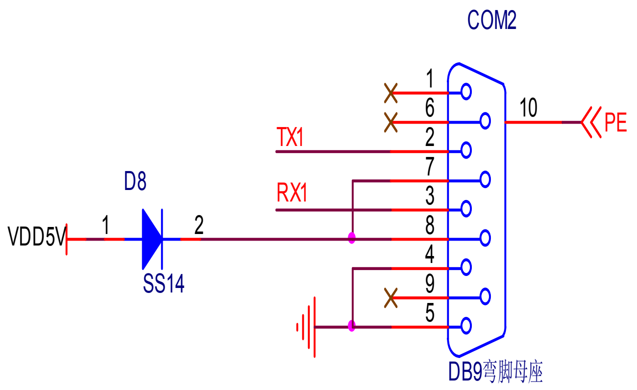


DB9 母头 DB9 female connector

Wire No.	Name	Function
1	NC	Null
2	TX1	Data transmitting
3	RX1	Data receiving
4	GND	Power ground
5	GND	Power ground
6	NC	Null
7	VDD5.0	Provide 5V power supply externally
8	VDD5.0	Provide 5V power supply externally
9	NC	Null

2.10.Serial port 2

Serial port 2 is used for R232-based Modbus protocol communication at present



DB9 弯脚母座 DB9 bent pin female socket

Wire No.	Name	Function
1	NC	Null
2	TX1	Data 1 receiving
3	RX1	Data 1 transmitting
4	GND	Power ground

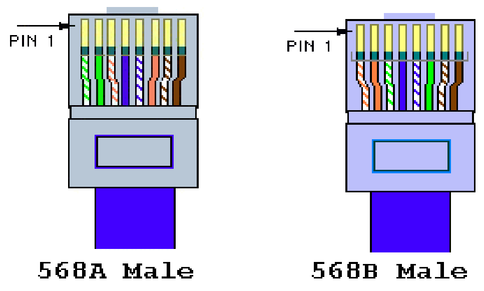
5	GND	Power ground
6	NC	Null
7	VDD5V	Provide 5V power supply externally
8	VDD5V	Provide 5V power supply externally
9	NC	Null

2.11.Standard USB interface (USB)

Standard USB storage (e.g. USB disk) interface

2.12.Standard Ethernet interface (RJ45)

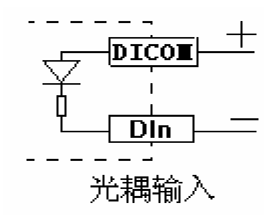
When the host is connected to the control card directly, please use crossover cable, i.e. use 568B category cable on one end and use 568A category cable on the other end.
When the host is connected to the control card through router, use 568A category cable on both ends.



2.13.Other reserved interfaces

Serial port 3, VGA interface, LVDS interface, SD card interface, and handheld box interface are reserved

2.14.Wiring mode of input signals

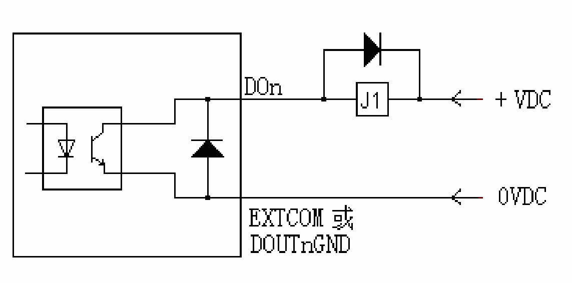


光耦输入 Optical coupling input

Connect INCOM terminal to positive end of external power supply, and connect the input signals to the pins of respective terminals.
In which, the common end of IN0—IN16 is INCOM1; the common end of IN17—IN33 is INCOM2; please connect the common end to +24V power supply. The input points are low voltage level valid. The current of single input shouldn't exceed 15mA and shouldn't be lower than 5mA.

2.15.Wiring mode of output signals

The switch quantity output of this control system is collector open circuit output. The common terminal is the first pin of JC1 and the GND of load power supply. Please connect pin-20 of JC1 to +24V power supply. The output point is low voltage level valid. Please connect the load between +24V and output point. The internal output circuit has relatively complete protection measures for over-current, overvoltage, short circuit, overheating and freewheeling; however, if inductive load is connected, e.g. relay, please connect freewheeling diodes on both ends of the relay coil, as shown below:



EXTCOM 或 DOUTnGND
EXTCOM or DOUTnGND

Note: The recommended voltage is lower than 24V and shouldn't exceed 30V. Do not short connect the anode and cathode, and do not short circuit the load, or else it will damage the module.

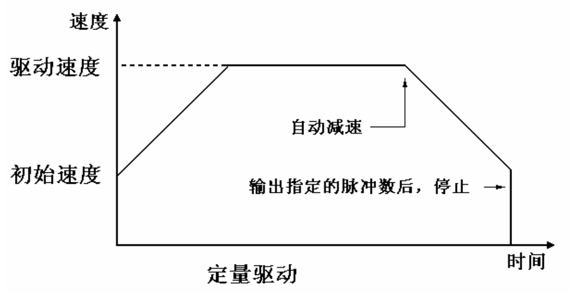
3. Motion Function

3.1. Quantitative drive

Quantitative driving is to output specified quantity of pulse in constant speed or acceleration/deceleration. Use this function to move to specified position or perform specific action. Quantitative driving of acceleration/deceleration is shown below. When the remaining of output pulse is less than acceleration accumulated pulses, it starts accelerating, and the driving also stops after outputting specified pulses.

The following parameters should be set for the quantitative driving for acceleration/ deceleration:

a) Acceleration/deceleration A/D
b) Start speed SV
c) Driving speed V
d) Output pulses P



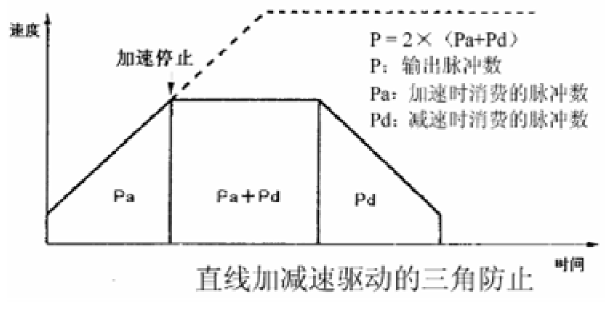
速度Speed　　驱动速度Driving speed
初始速度Start speed　　自动减速 Automatic deceleration
输出指定的脉冲冲数后，停止 Stop after outputting specified pulses
定量驱动Quantitative driving　　时间 Time

Acceleration/deceleration quantitative driving usually starts automatic acceleration from the calculated deceleration point shown in the picture above.

Triangle prevention of quantitative driving

During quantitative driving of linear acceleration/deceleration, if output pulses are lower than required pulses, triangle wave will be generated as in the picture above, and the triangle prevention function will be enabled at this moment.

The triangle prevention function also prevents triangle wave even if the output pulses are less than required pulses in quantitative driving of linear acceleration/ deceleration; during accelerating, when the consumed pulses are more than 1/2 of total output pulses in accelerating and decelerating, it stops accelerating and keeps at constant speed. Therefore, even if the output pulses are less than 1/2 of total output pulses, it is still in constant speed.



速度Speed
加速停止Accelerating stop
P：输出脉冲数 P: Output pulses
Pa ：加速时消费的脉冲数
Pa: Pulses consumed in accelerating
Pd ：减速时消费的脉冲数
Pd: Pulses consumed in decelerating
直线加减速驱动的三角防止
Triangle prevention driven by linear acceleration/deceleration
时间Time

3.2. Continuous drive

During continuous driving, output driving pulse continuously until high level stop command or external stop signal is valid. Use this function for home search and motor rotation control.

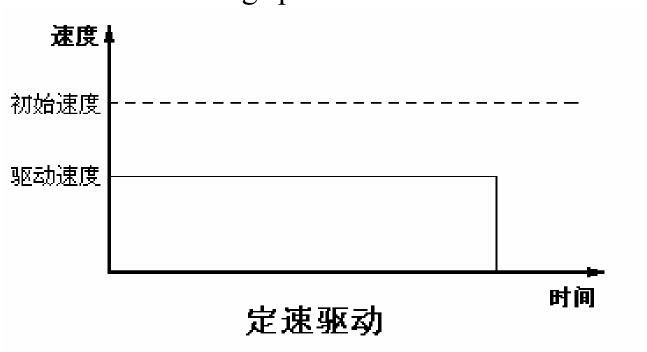
3.3. Speed curve

The driving pulse output of every axis usually uses positive/negative quantitative driving command or continuous driving command. In addition, the speed curves of constant speed, linear acceleration/deceleration and S curve acceleration/deceleration are generated by setting mode or parameters.

3.4. Constant speed drive

Constant speed driving will output driving pulse in constant speed. If the driving speed is lower than the start speed, there will be constant speed driving only instead of acceleration/deceleration driving. When use home search, encoder Z phase and similar signals, acceleration/deceleration driving isn't required if stop immediately after signal is searched; instead, the system runs low speed constant driving.
The following parameters should be set for constant speed driving:

- Start speed SV
- Driving speed V

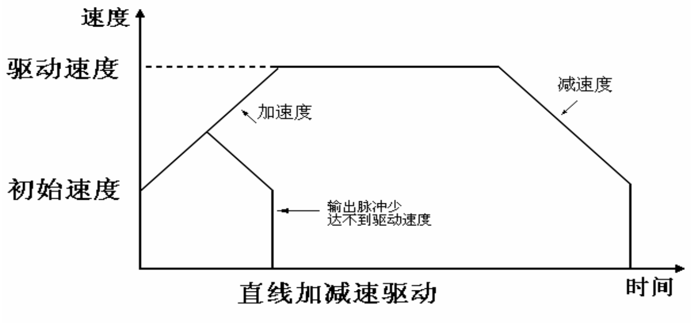


速度	Speed
初始速度	Start speed
驱动速度	Driving speed
定速驱动	Constant speed driving
时间	Time

3.5. Linear acceleration/deceleration drive

Linear acceleration/deceleration is to accelerate from the start speed to specified driving speed linearly.
During quantitative driving, the acceleration counter records the accumulated pulses. When the remaining output pulses are less than acceleration pulse, it starts decelerating (automatic deceleration), and decelerates to start speed linearly in specified deceleration.
The following parameters should be set for linear acceleration/deceleration driving:

- Acceleration A; acceleration and deceleration
- Deceleration D; the deceleration when acceleration and deceleration are set separately (if necessary)
- Start speed SV
- Driving speed V
- Output pulses P; used in quantitative driving (if necessary)



速度 Speed
驱动速度 Driving speed
加速度 Acceleration
减速度 Deceleration
初始速度 Start speed
输出脉冲少达不到驱动速度
Output pulse is low and can't reach driving speed
直线加减速驱动
Linear acceleration/deceleration driving
时间 Time

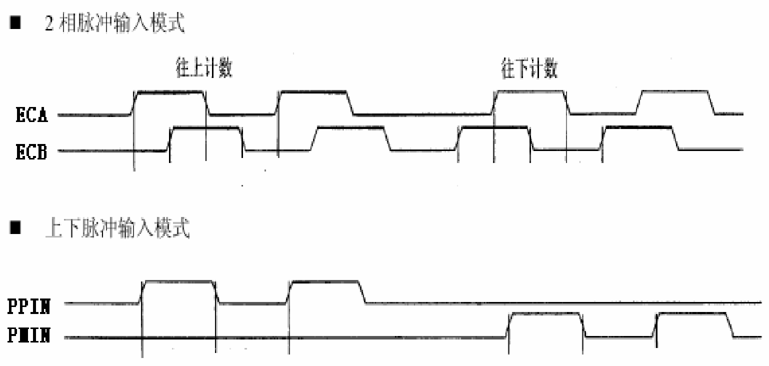
3.6. Look-ahead algorithm and S curve acceleration/deceleration drive

During continuous track processing, the user can enable the look-ahead algorithm integrated in PMC. This algorithm can check the corner size or ending of the processing track ahead, and adjust the feed speed automatically to avoid the system having frequent start-stop at the connection of segments. It can significantly improve the processing efficiency, smooth processing speed, prevent processing impact and lower user's workload on processing speed control. In this mode, the user also can select linear or S curve acceleration/deceleration mode.

3.7. Position management

◆ **Logical position counter and actual position counter**

The logical position counter outputs pulse in positive/negative counting direction, counts up 1 when outputs a positive pulse and counts down 1 when outputs a negative pulse.
The actual position counter counts the input pulses from external encoder, allows selecting A/B phase signal or independent 2 pulses up/down counting signals for input pulses with command, and the counting direction can be set.
Allow writing or reading the data of two counters, and the counting range is -2,147,483,648~+2,147,483,647



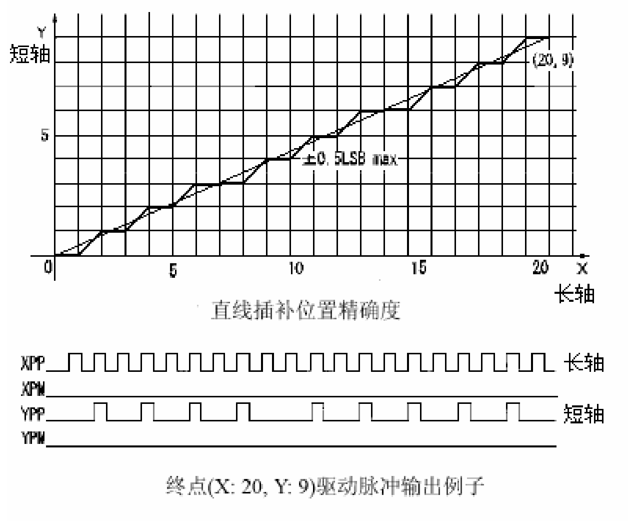
2相脉冲输入模式 Two-phase pulse input mode
往上计数 Count up 往下计数 Count down
上下脉冲输入模式 Up/down pulse input mode

3.8. Interpolation

The controller allows 2-6 axes linear interpolation.
During interpolation driving, the interpolation calculation is run under the basic pulse sequence of specified X axis, and therefore, before interpolation command, set the start speed and driving speed of specified X axis first.

3.8.1 2-6 axes linear interpolation

The linear interpolation starts after setting the end point coordinates relative to current position. The coordinate range of linear interpolation is 24-bit with symbol, and the interpolation range is from current position of every axis to -8,388,607~+8,388,607.



短袖 Short axis
直线插补位置精确度
Linear interpolation position precision
长袖 Long axis
长袖 Long axis
长袖 Short axis
终点（X:20,Y:9）驱动脉冲输出例子
Example of end point (X: 20, Y: 9) driving pulse output

As shown in the picture above, the position precision of specified straight line is ±0.5LSB in the entire interpolation range. Above picture also has an example of driving pulse output of linear interpolation. In the set end point value, the axis with maximum absolute value is long axis, which always outputs pulse during interpolation driving, and others are short axes. According to the result of linear interpolation arithmetic, sometimes output pulse, and sometimes do not output pulse.

3.8.2 Hardware interpolation cache

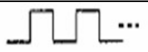
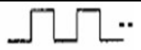
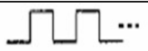
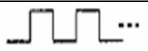
For the control cards without hardware interpolation function, to continue next interpolation after previous interpolation is ended, the device has to continuously check whether previous interpolation completes and then output the data of next interpolation. If the upper computer is very slow, or running multi-task operating system, two interpolations will have pause, which will affect the interpolation and also can’t improve the interpolation speed.
The hardware interpolation cache provided by the control card can solve this problem. It can save multiple interpolation commands in hardware cache. It can write the interpolation command even while executing interpolation command motion.
When command is written in empty hardware cache, the control card will execute thee first read-in command immediately, and follow the principle of first read-in first execution. When hardware cache is empty, the controller stops automatically after executing current interpolation motion. To save interpolation command in hardware cache, it is necessary to check whether the cache is full. If yes, the interpolation commands can’t be written in, or else the commands will be lost.

The hardware interpolation cache can effectively prevent the pause between two interpolations, and have better effect even if the computer is very slow.

3.9. Pulse output mode

Driving output pulse has the following two pulse output modes below. When independent two pulses mode is used, PU/CW outputs driving pulse in positive driving, and DR/CCW outputs driving pulse in negative driving. When one pulse mode is used, PU/CW outputs driving pulse and DR/CCW outputs direction signal.

Both pulse and direction are positive logic setting

Pulse output mode	Driving direction	Output signal wave	
		PU/CW signal	DR/CCW signal
Independent two pulses mode	+ direction driving output		Low level
	- direction driving output	Low level	
One pulse mode	+ direction driving output		Low level
	- direction driving output		High level

3.10. Hardware limit signal

Hardware limit signal (LMT+, LMT-) is the input signal that limits positive and negative driving pulse. When the logical voltage level and limit signal are valid, the motion of the axis will be stopped immediately.

3.11. Servo motor corresponding signal

The input signals connected to servo motor drive are in-position signal (INPOS) and alarm signal (ALARM). The valid/invalid and logical level of every signal can be set.
INPOS input signal corresponds to the servo motor positioning complete signal. If the mode is set to valid, wait INPOS input signal valid after one driving ends, the input signal is valid, the driving state returns and ends. ALARM input signal receives alarm signal from servo motor drive. If it is set to valid, it monitors ALARM input signal, and stops driving immediately if the signal is valid. These input signals used for servo motor drive can be read with universal I/O function. Universal output signals can be used for bias counter clear, alarm reset, servo on, etc.

4. PMC System Architecture

ADT-8860 is a multipurpose and onsite programmable six axes motion controller (PMC -- program motion control) with PLC function. It supports offline operating mode, online real-time operating mode and console operating mode, and is widely used in various automation control equipment.

Offline operating mode

When the controller is running in offline mode, it used C similar language style instruction programming and control, and is partly compatible with G code motion instructions. The programming is simple, flexible, support multi-task mode and can achieve complicated calculation and control.

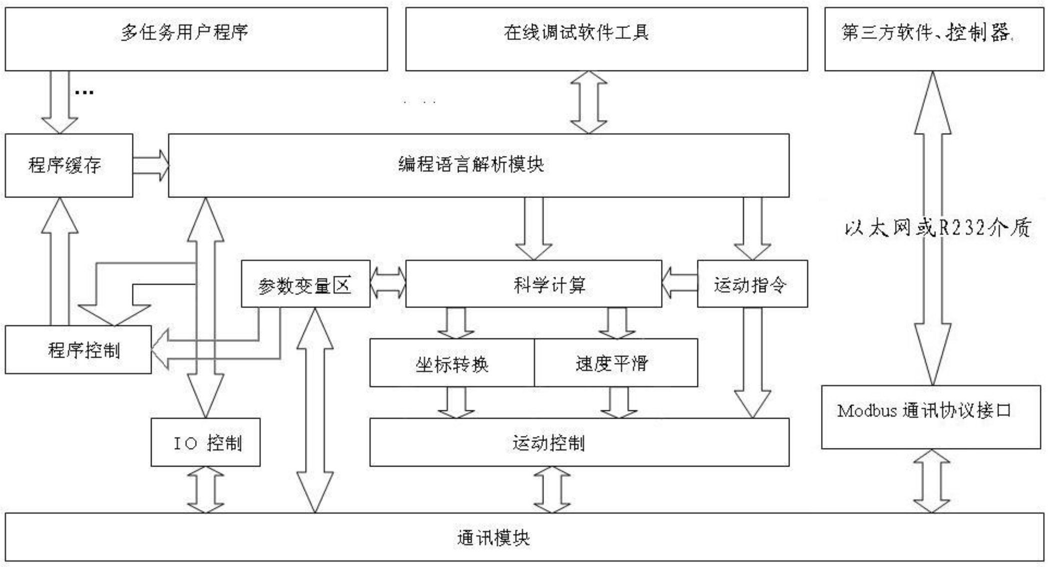
Online operating mode

When it is in online running mode, any third party software can access PMC internal system register area through standard Modbus communication protocol to control the various function of the motion control card and provide flexible open architecture.

Console operating mode

In the process of application development, the user can control the whole process of PMC through serial communication based console and get desired system info, which is helpful to program testing.

The framework diagram of PMC system follows:



多任务用户程序 Multi-task user program　　在线调试软件工具 Online testing software tool
第三方软件、控制器 Third party software, controller　　程序缓存 Program cache
编程语言解析模块 Programming language analysis module　　以太网或 R232 介质 Ethernet or RS232 media
参数变量区 Parameter variable area　　科学计算 Scientific calculation
运动指令 Motion instruction　　程序控制 Program control　　坐标转换 Coordinates conversion
速度平滑 Speed smoothing　　IO 控制 IO control　　运动控制 Motion control
Modbus 通讯协议接口 Modbus communication protocol interface　　通讯模块 Communication module

PMC features and introduction

PMC controller provides 32 independently running task spaces for the user. The task No. and priority are 0-31, and the lower number has higher priority. Every task program can run independently or through IO signal, while the internal variables or upper computer controller run multi-task. The user only needs to use the instruction function provided by the PMC system to write different applications according to the specific requirement of product process, and then download to PMC with development software tool to test and run.

The instruction function provided by the PMC system includes most of the operation expressions, condition execution, cycle, goto, subroutine call, return and other commonly used program execution structure in C language, and also increases motion control related instructions and function according to the technical characteristics of motion control industry, e.g. quick shift instruction, linear interpolation instruction, arc/helix interpolation instruction, read/write IO instruction, logic axis mapping, coordinate system conversion, look-ahead and smoothing, pause and resume, software/hardware limit, motion status query, motion conflict treatment mechanism, delay waiting, auto-home, system configuration instructions and other advanced functions, so that the user get rid of hardware design and low-level driver design when developing own control system, and only needs to write related control programs according to process requirement. It will help the customers to simplify system design, shorten system development period, reduce development cost, and improve system maintenance performance.

The user programs written with C script language and instruction function provided by PMC can be classified into startup programs, IO scanning programs, PLC programs, and motion control programs according to the function. The programs may cross over and aren’t classified strictly. The crossover of multiple or different types of user programs can achieve complicated system function. In addition, the simultaneous running of multi-task can help users to save required controller quantity in large control system.

Most instructions provided in PMC can be entered and executed directly in serial port console. Console is a command line based human-computer interaction mode, and any standard serial I/O software can be used as running interface of the console, regardless Windows, Linux or any other operating systems. Through the console, the user can fully operate and monitor the system status in the process of program development, improving the system testing efficiency significantly.

PMC system parameter area and open characteristics

PMC also provides various register areas for system configuration, function setting and calculation. The console can access these register areas directly, and these registers are open to third party software through Modbus communication protocol. PMC supports serial R232 based and Ethernet based Modbus protocols at present, both of which support RTU and ASCII transmission mode, while the Ethernet based Modbus protocol also supports TCP/IP or UTP protocol based mode. By opening these register areas of special functions, PMC can be controlled by other controllers or software. The PMC development kit CD provided by Adtech contains Modbus communication protocol based VC and VB upper computer development kit, combines the functions based on Modbus register operation, and re-package into API functions that can be called in VC and VB to facilitate the online control and secondary development for users used to PC based control. The user also can customize and package advanced API functions through most basic Modbus protocol register operation based low-level function to achieve more complicated function control. Similarly, the user also can use Modbus communication protocol supported and configuration software based touch screen controller to control PMC online. Before this, these touch screen controllers are widely used in PLC controller industry. Through opening these register areas, PMC achieves more flexible system expansion characteristics, realizes controller network group consisting of several PMC through Ethernet based Modbus communication protocol, and expands the application of PMC in plant-level controller network system.

5. System Register and MODBUS Protocol

PMC register introduction

ATD-8860 PMC motion controller provides several register areas of special functions, used to configure basic system parameters or participate in the calculation and control process. Most register areas can use third party software or controller to access through Modbus communication protocol, and ensure the perfect openness and scalability of the system.

The system registers consist of I register area, E register area, S register area, C register area, D register area, F register area, and L register area, of which I, E, S, C, D, and F register areas are global variables; L register area is internal variable of user program, and can’t be accessed by external program. The register areas are described below.

I register area: input (global)

32bit unsigned integer data. The addressing range of the register is I0-I1023. This register area also supports bit addressing and the addressing range is 0-32767, of which the lower 0-79 bits corresponds to the real physical input IO of the controller, and other higher addressing bits are used as virtual input IO in the program. The word access function code of I register area in Modbus is x04 (read multiple input registers), the address range is 0-2046, and two words (16bit) are combined into one unsigned integer (32bit); Modbus input point access function code is x02 (read multiple input points state), and the address range is 0—32767. I register area is usually used to process signal input and participate in the calculation or control process.

E register area: output (global)

32bit unsigned integer data. The addressing range of the register is E0-E8191. This register area also supports bit addressing and the addressing range is 0-65535, of which the lower 0-23 bits corresponds to the real physical input IO of the controller, and other higher addressing bits are used as common bit registers in the program. The word access function code of E register area in Modbus is x03 (read holding register), x10 (write multiple holding registers), the address range is 0-16382, and two words (16bit) are combined into one unsigned integer (32bit); Modbus output point access function code is x01 (read multiple output points state), x05(write single output point state) and x0f(read multiple output points state), and the address range is 0—32767. E register area is usually used to process signal input and participate in the calculation or control process.

S **register area: signed integer (global)**

32bit signed integer data. The addressing range of the register is S0-S8191, and doesn’t support bit addressing. The word access function code of S register area in Modbus is x03 (read holding register), x10 (write multiple holding registers), the address range is 16384-32766, and two words (16bit) are combined into one unsigned integer (32bit). S register area is usually used to process calculation or control process.

C register area: system configuration (global)

32bit unsigned integer data. The addressing range of the register is C0-C2047, and doesn’t support bit addressing. The word access function code of C register area in Modbus is x03 (read holding register), x10 (write multiple holding registers), the address range is 32768-36862, and two words (16bit) are combined into one unsigned integer (32bit); C register area is usually used to configure system parameters, e.g. serial port configuration info, network interface info, startup options, system running info, special read/write area, etc. Other registers are reserved. The registers in C register area are described below:

C0: serial port 0 baud rate, 9600/14400/19200/38400/56000/57600/115200 optional, default: 115200

C1: serial port 0 data bit, fixed at 8 (read-only)

C2: serial port 0 stop bit, 1 bit/2-bit optional, default: 1

C3: serial port 0 checksum mode, 0=None/1=Odd/2=Even optional, default: 0

C4: serial port 0, 0 – hide testing info, 1 – show program execution process, 2 — show testing info

C5-C9: reserved

C10: card No. address, 0-127, default: 0

C11: serial port 1 baud rate, 9600/14400/19200/38400/56000/57600/115200 optional, default: 115200

C12: serial port 1 data bit, fixed at 8 (read-only)

C13: serial port 1 stop bit, 1 bit/2-bit optional, for Modbus-RTU protocol, select 2

C14: serial port 1 checksum mode, 0=None/1=Odd/2=Even optional, default: 0

C15-C19: reserved for serial port 3

C20-C39: reserved

C40: IP address, default: 192.168.0.123

C41: IP address mask, default: 255.255.255.0

C42: gateway IP address, default: 192.168.0.1

C43-C44: NIC physical address, default: 00-AB-CD-00-12-03

C45: network communication port No., default: registered port No. of Modbus: 502

C46-C49 reserved

C50-C59: second network interface parameter area, reserved

C60: quantity of current used coordinate systems, (read-only)

C61: mask of used coordinate system No., (read-only)

C62: quantity of currently running programs, (read-only)

C63: mask of currently running program No., (read-only)

C64: quantity of currently opened programs, (read-only)

C65: mask of currently opened program No., (read-only)

C66: No. of currently running program

C67-C69 reserved

C70: MODBUS IO read/write mode, 0 - read/write IO cache, 1 - read/write real IO and refresh cache

C71: load specified program after startup, 0 - No, 1 - Yes

C72: run specified program after startup, 0 - No, 1 - Yes

C73-C76 specify startup file name, 8+3 file name format, default file name is “START.BAT”

C77: write 1 — load and run the startup file

C79: write 1 — load system parameters from power failure memory, write 2 -- load system parameters from file system,

C80: write 1 — save system parameters in power failure memory, write 2 -- save system parameters in file system

C96: alarm state word 1: No. of task that has error

C97: alarm state word 2: position of error code

C98: alarm state word 3: error code

C99-C1023: reserved

C1024-C1535: Modbus special write function area, used to realize special function instructions that require writing multiple registers of discrete address simultaneously and improve the instruction efficiency
C1536-C2013: Modbus special read function area, used to realize special function instructions that require reading multiple registers of discrete address simultaneously and improve the instruction efficiency

D register area: motion (global)
32bit unsigned integer data. The addressing range of the register is D0-D2047, and doesn't support bit addressing. The word access function code of D register area in Modbus is x03 (read holding register), x10 (write multiple holding registers), the address range is 36864-40958, and two words (16bit) are combined into one unsigned integer (32bit). D register area is usually used to configure motion parameters, of which D0-D99 are used to set global motion parameters, and Dx00-Dx99 are used to set motion parameters of every axis (1<= x<=6). Other registers are reserved. The registers in D register area are described below:
D0: home status
D1: system interpolation status, 0: idle, 1: interpolating
D2: system motion status, 0: idle, 1: in motion
D3: feed mode, 0--dimension unit/unit time, 1 -- dimension unit/principal axis rotation
D4: in MODBUS communication protocol or console mode, the user specifies desired coordinate system; if the user operates this register in program task, the current coordinate system of the task is set.
D5: specify the task for info view or operation
D6: view the No. of current coordinate system in specified task in D5 register (read-only)
D7: pretreatment motion segment in look-ahead; the motion starts when reaching this quantity or completing all
D32: global motion stop signal, 0: normal, 1: immediate stop, 2: decelerating stop, and then stop running task and current failed task, and specify the programs to be stopped in advance
D33: global motion pause signal, 0: normal, 1: pause
D34: dimension unit, 0: mm, 1: inch, default: 0
D35: time unit; 0: sec, 1: min, default: 1
D36: current coordinate mode of current task, 0: increment coordinate mode, 1: absolute coordinate mode; please note that the coordinate mode of the coordinate system specified by every task is independent, and only the coordinate mode of current coordinate system in the task is operated when any task reads/writes this parameter.
D37: task motion status: 0: the motion control isn't solely held by certain task, and any task can execute relation motion instructions immediately when the system is idle; non-0: the motion control of the PMC has been held by certain task solely, its value = task No. +1; the sole holding state can be cleared by the task or user command only.
D38: Multitask axis motion conflict treatment; 0: pause and report error, 1: execute new motion when the system is in idle, 2: if there is no axis conflict in non-interpolation state, allow executing multi-axis linkage again
D39: Motion command execution mode, 0 -- execute next motion command only after execution of the current motion command in real-time, and each motion command has independent acceleration and deceleration segments; 1—send continuous motion commands to hardware cache, and then the hardware executes continuous interpolation motion; each motion command is uniform motion, and doesn't have acceleration and

deceleration segments; 2—implement speed forward smoothing for continuous motion command according to its motion track; default: 0

D40: Feed starting speed	(dimension unit/unit time)
D41: Feed operating speed	(dimension unit/unit time)
D42: Maximum feed speed	(dimension unit/unit time)
D43-D45: reserved	
D46: Linear acceleration	(dimension unit/unit time^2)
D47: S variable acceleration time	(ms)
D48: S variable acceleration	(read-only); this register is calculated according to D47 and D48 (dimension unit/unit time^2)
Dx00: reserved	
Dx01: Axis pulses per rotation	(pulse/rotation)
Dx02: Encoder lines	(line/rotation)
Dx03: Axis pulse equivalent – numerator	(pulse)
Dx04: Axis pulse equivalent – denominator	(dimension unit)
Dx05: Axis home starting speed	(dimension unit/unit time)
Dx06: Axis home maximum speed	(dimension unit/unit time)
Dx07: Axis home scanning speed	(dimension unit/unit time)
Dx08-Dx10: reserved	
Dx11: Axis home acceleration	(dimension unit/unit time^2)
Dx12: reserved	
Dx13: Axis home maximum stroke	(dimension unit)
Dx14: Axis home scanning stroke	(dimension unit)
Dx15: Axis starting operating speed	(dimension unit/unit time)
Dx16: Axis maximum operating speed	(dimension unit/unit time)
Dx17: Axis starting operating speed	(rotation/unit time)
Dx18: Axis maximum operating speed	(rotation/unit time)
Dx19: Linear acceleration	(dimension unit/unit time^2)
Dx20-DX24: reserved	
Dx25: Position error limit – alarm	(pulse)
Dx26-Dx28: reserved	
Dx29: Deceleration time of abnormal stop	(ms)

Dx30: Hardware positive/negative limit signal setting

- 2bit—limit signal level, 0: low, 1: high
- 1bit—0: negative limit valid, 1: negative limit invalid
- 0bit—0: positive limit valid, 1: positive limit invalid

Dx31: Hardware home signal; stop immediately if effective, or else the software executes

- 1bit— set effective signal lvel, 0: low, 1: high
- 0bit—0: Hardware home signal invalid, 1: Hardware home signal valid

Dx32: Hardware Z phase signal valid; if valid, stop when encounters Z phase signal in home scanning process;

- 1bit— set effective level of hardware Z phase signal, 0: low, 1: high
- 0bit—0: Hardware Z phase signal invalid, 1: Hardware Z phase signal valid

Dx33: Home step

Dx34: If the logical position of the axis is near zero point or home, and is executing home motion, the axis first moves the millimeters forward at half speed

Dx35: When the axis is several unit distance from the home of logical position and is executing home motion, the axis first moves the distance set by Dx34 forward at half speed, and then executes the home motion.

Dx36: Pulse mode setting

- 2bit- pulse mode setting, 0: pulse + pulse, 1: pulse + direction
- 1bit- pulse logic setting, 0: positive logic pulse, 1: negative logic pulse
- 0bit- pulse direction logic setting, 0: direction outputs positive logic, 1: direction outputs negative logic

Dx37: reserved

Dx38: Get axis driven state (read-only), 0 = no drive, 1 = driving

Dx39-Dx46: reserved

Dx47: Get axis feed speed

Dx48: Get axis logical position of machine tool coordinate system (pulse)

Dx49: Get axis actual position of machine tool coordinate system (line)

Dx50: Get or set the axis logical position of the coordinate system specified in D4 register, and the value of Dx51 register will be changed synchronously when setting this register. Multiple coordinate systems share this register address in virtual mode

Dx51: Get or set the axis offset position of the coordinate system specified in D4 register, and the value of Dx50 register will be changed synchronously when setting this register. Multiple coordinate systems share this register address in virtual mode

F register area: double-precision floating point (global)

64bit double-precision floating point data. The addressing range of the register is F0-F4095, and doesn't support bit addressing. The word access function code of F register area in Modbus is x03 (read holding register), x10 (write multiple holding registers), the address range is 40960-49148, and four words (16bit) are combined into one double-precision floating point (64bit). F register area is usually used to process calculation or control process.

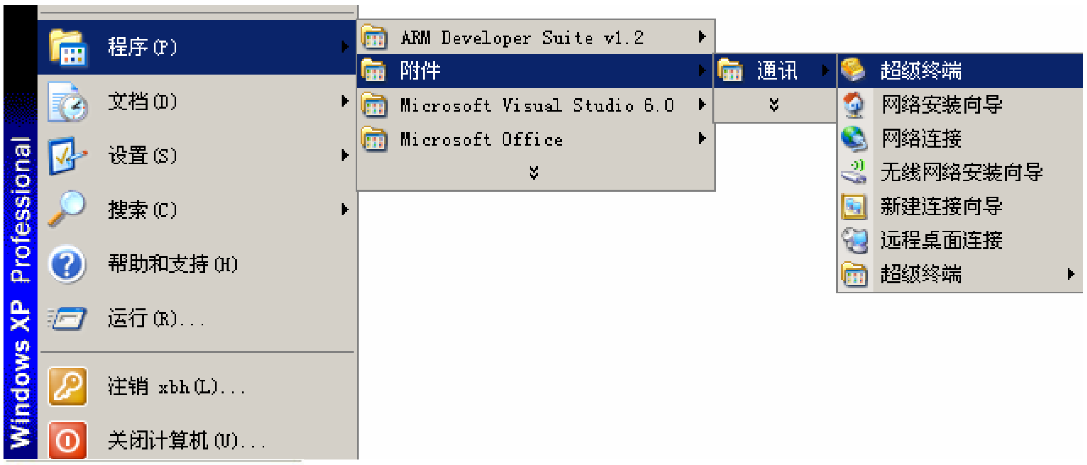
L register area: internal variables (global)

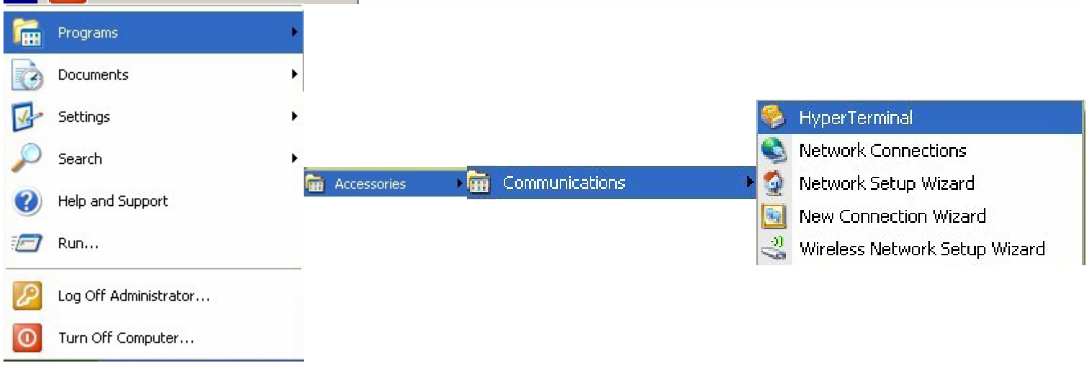
32bit signed integer data, and doesn't support bit addressing. The addressing range of the register is L0-L1023, and every task the register independently. L register area can't be accessed in Modbus, because it is private register area of every task. L register area is usually used to process calculation or control process.

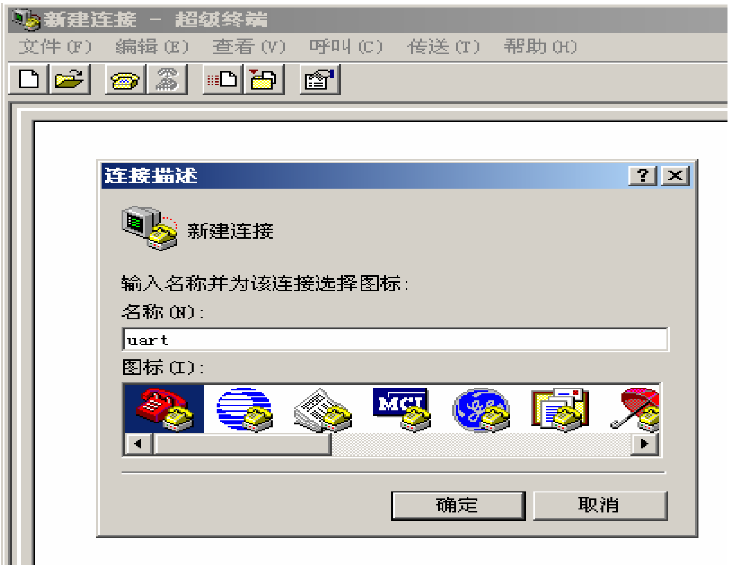
6. Console Function Description

Console is a serial communication and command line based basic human-computer interaction mode. The consoled bases on serial RS232 communication media and can be run in any serial testing software, e.g. the Hyper Terminal in Windows. Through the console, the user can fully operate and monitor the system status in the process of program development, improving the system testing efficiency significantly. The SSCOM32.EXE serial testing tool provided in this system CD even can be used in Linux.

The following screenshots describe how to create Hyper Terminal in windows and set the parameters.






新建连接 - 超级终端

文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)

连接描述

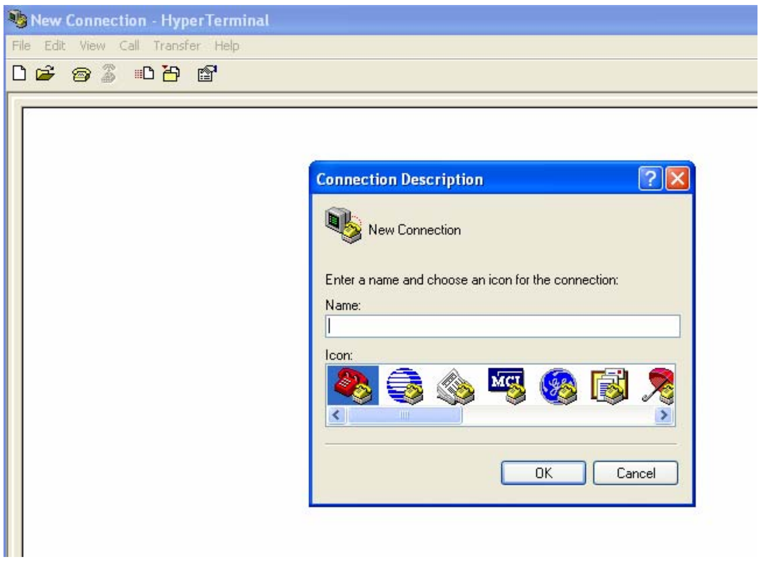
新建连接

输入名称并为该连接选择图标:

名称(N): uart

图标(I):

确定 取消


New Connection - HyperTerminal

File Edit View Call Transfer Help

Connection Description

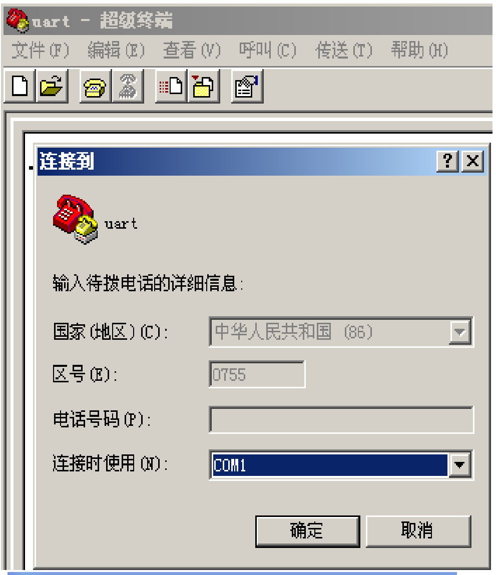
New Connection

Enter a name and choose an icon for the connection:

Name:

Icon:

OK Cancel


uart - 超级终端

文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)

连接到

uart

输入待拨电话的详细信息:

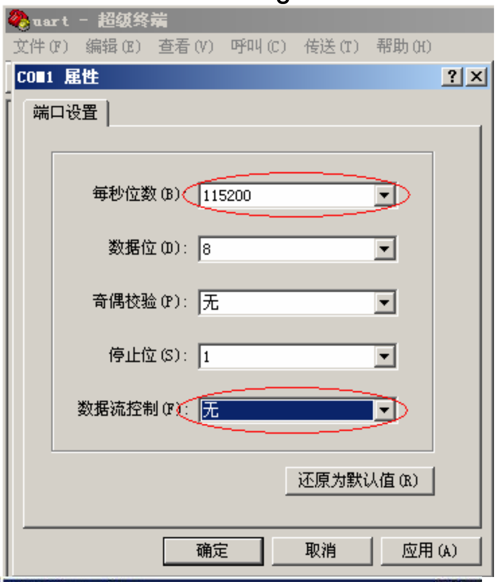
国家(地区)(C): 中华人民共和国 (86)

区号(E): 0755

电话号码(P):

连接时使用(N): COM1

确定 取消


uart - 超级终端

文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)

COM1 属性

端口设置

每秒位数(B): 115200

数据位(D): 8

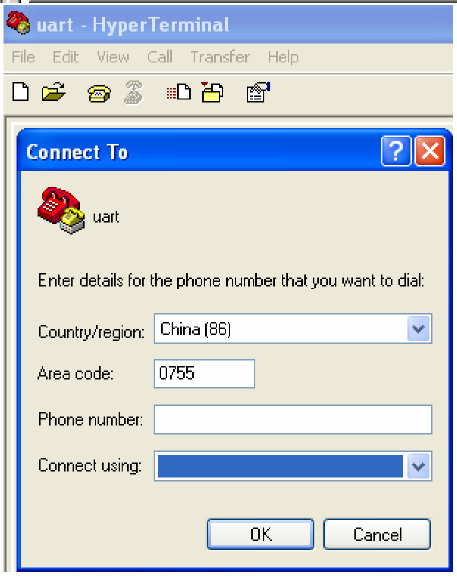
奇偶校验(P): 无

停止位(S): 1

数据流控制(F): 无

还原为默认值(R)

确定 取消 应用(A)


uart - HyperTerminal

File Edit View Call Transfer Help

Connect To

uart

Enter details for the phone number that you want to dial:

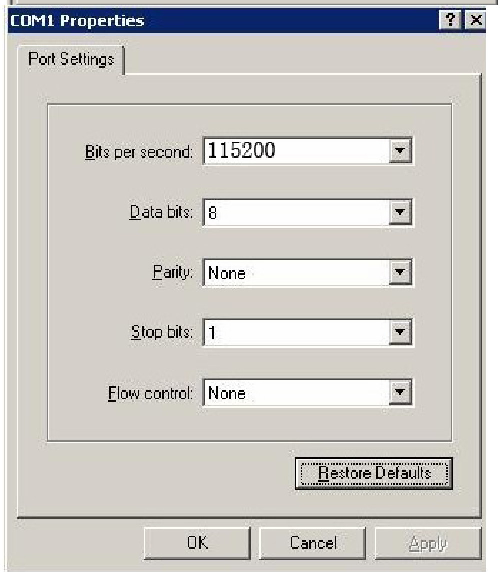
Country/region: China (86)

Area code: 0755

Phone number:

Connect using:

OK Cancel


COM1 Properties

Port Settings

Bits per second: 115200

Data bits: 8

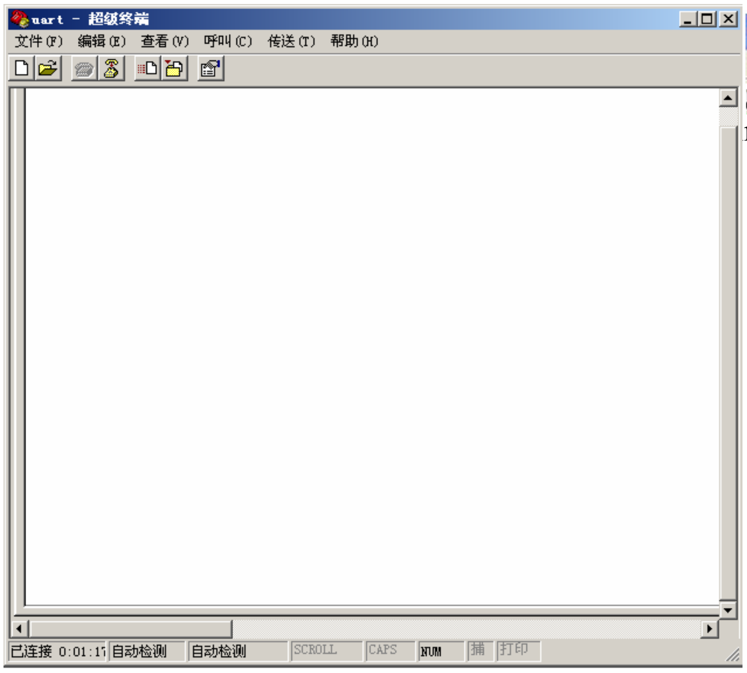
Parity: None

Stop bits: 1

Flow control: None

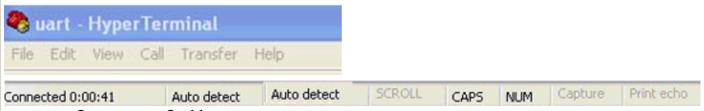
Restore Defaults

OK Cancel Apply


uart - 超级终端

文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)

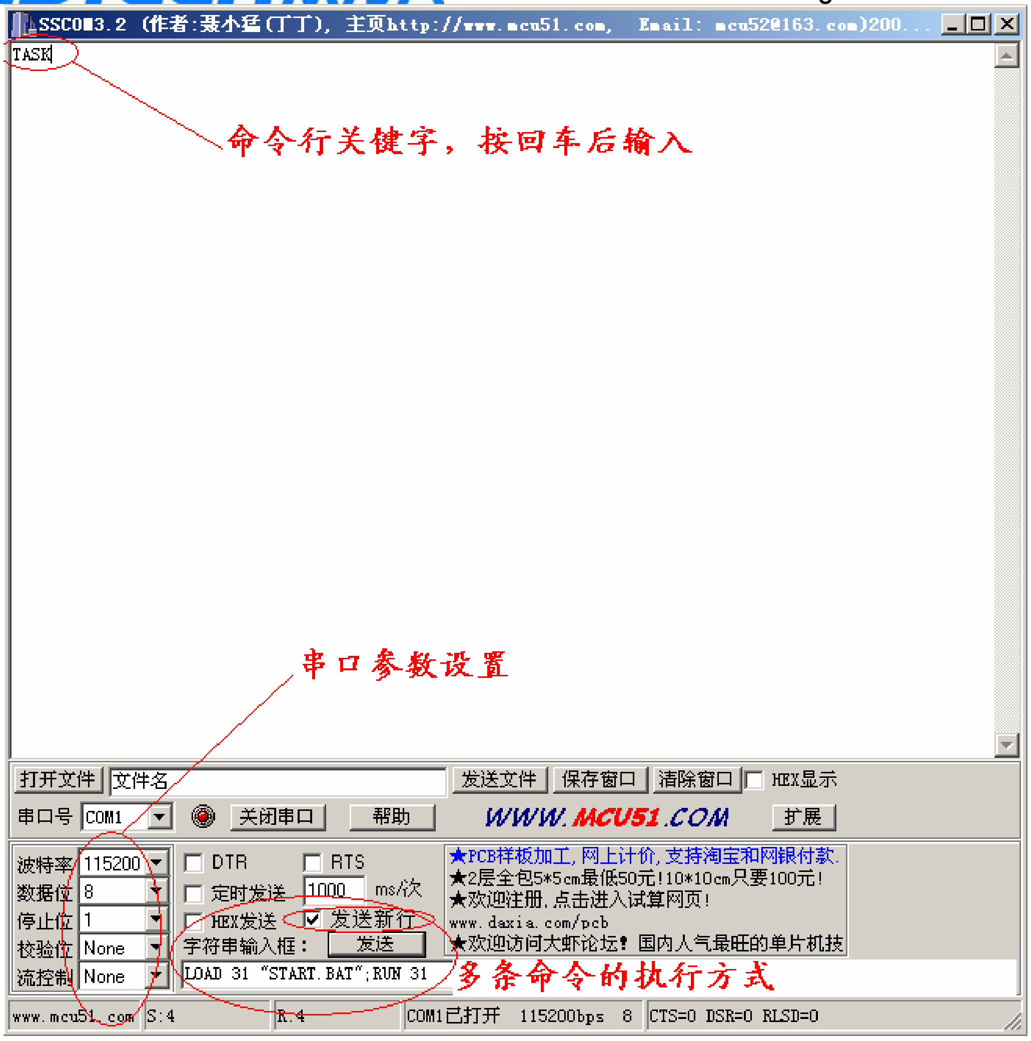
已连接 0:01:17 自动检测 自动检测 SCROLL CAPS NUM 捕 打印


uart - HyperTerminal

File Edit View Call Transfer Help

Connected 0:00:41 Auto detect Auto detect SCROLL CAPS NUM Capture Print echo

ng software follows:



SSCOM3.2 (作者:聂小猛(丁丁), 主页http://www.mcu51.com, Email: mcu52@163.com)200...

```
TASK
```

命令行关键字，按回车后输入

串口参数设置

多条命令的执行方式

串口号 COM1
波特率 115200
数据位 8
停止位 1
校验位 None
流控制 None
定时发送 1000 ms/次
[✓ 发送新行]

```
LOAD 31 "START.BAT";RUN 31
```

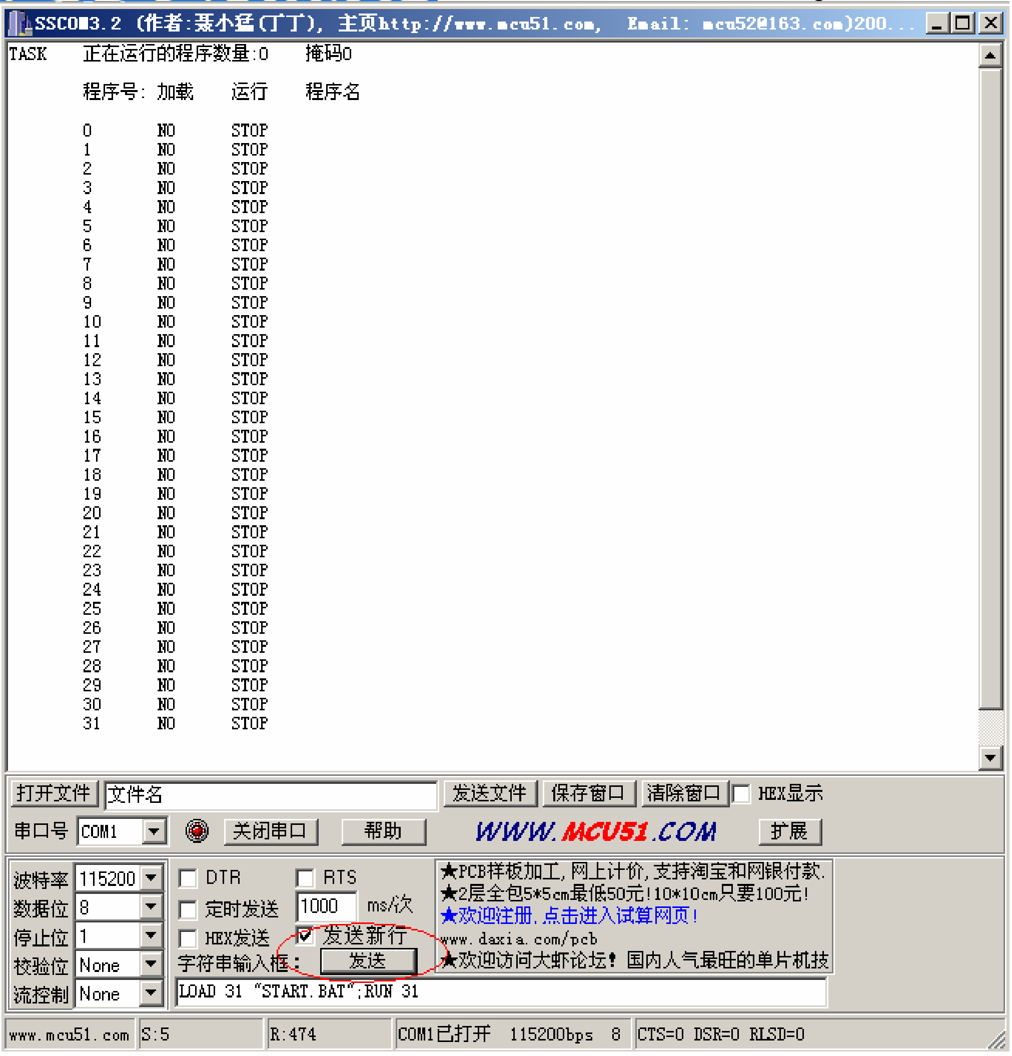
★PCB样板加工，网上计价，支持淘宝和网银付款.
★2层全包5*5cm最低50元！10*10cm只要100元！
★欢迎注册，点击进入试算网页！
www.daxia.com/pcb
★欢迎访问大虾论坛！国内人气最旺的单片机技

```
www.mcu51.com  S:4  R:4  COM1已打开  115200bps  8  CTS=0 DSR=0 RLSD=0
```

命令行关键字，按回车后输入 Command line keywords; press Enter the insert
串口参数设置 Serial parameter settings
多条命令的执行方式 Execution mode of multiple commands
打开文件/文件名 Open file: File name
串口号 Serial port No. \关闭串口 Close serial port \ 帮助 Help
发送文件 Send file \ 保存窗口 Save window \ 清除窗口 Clear window \HEX 显示 HEX display
扩展 Extend / 波特率 Baud rate / 数据位 Data bit / 停止位 Stop bit / 检验位 Parity / 流控位 Flow control
定时发送 Schedule send / HEX 发送 HEX send / 字符串输入框 String entry box
发送新行 Send new line
发送 Send

In the console, the user can get desired system info or operate the control system directly by entering related commands, e.g. read/write variables in every register area of the system, load/start/stop programs, view system and running info, set system operating mode, etc. It even supports common mathematical expressions and instruction help info, and the powerful console function is helpful to user in testing program and controlling system status.

After executing TASK command in above picture, the interface follows:



SSCOM3.2 (作者:聂小猛(丁丁), 主页http://www.mcu51.com, Email: mcu52@163.com)200...

```
TASK      正在运行的程序数量:0     掩码0

          程序号: 加载     运行     程序名

          0       NO       STOP
          1       NO       STOP
          2       NO       STOP
          3       NO       STOP
          4       NO       STOP
          5       NO       STOP
          6       NO       STOP
          7       NO       STOP
          8       NO       STOP
          9       NO       STOP
          10      NO       STOP
          11      NO       STOP
          12      NO       STOP
          13      NO       STOP
          14      NO       STOP
          15      NO       STOP
          16      NO       STOP
          17      NO       STOP
          18      NO       STOP
          19      NO       STOP
          20      NO       STOP
          21      NO       STOP
          22      NO       STOP
          23      NO       STOP
          24      NO       STOP
          25      NO       STOP
          26      NO       STOP
          27      NO       STOP
          28      NO       STOP
          29      NO       STOP
          30      NO       STOP
          31      NO       STOP
```

打开文件 文件名 发送文件 保存窗口 清除窗口 HEX显示

串口号 COM1 关闭串口 帮助 WWW.MCU51.COM 扩展

波特率 115200 DTR RTS
数据位 8 定时发送 1000 ms/次
停止位 1 HEX发送 ✓ 发送新行
校验位 None 字符串输入框: 发送
流控制 None

LOAD 31 "START.BAT";RUN 31

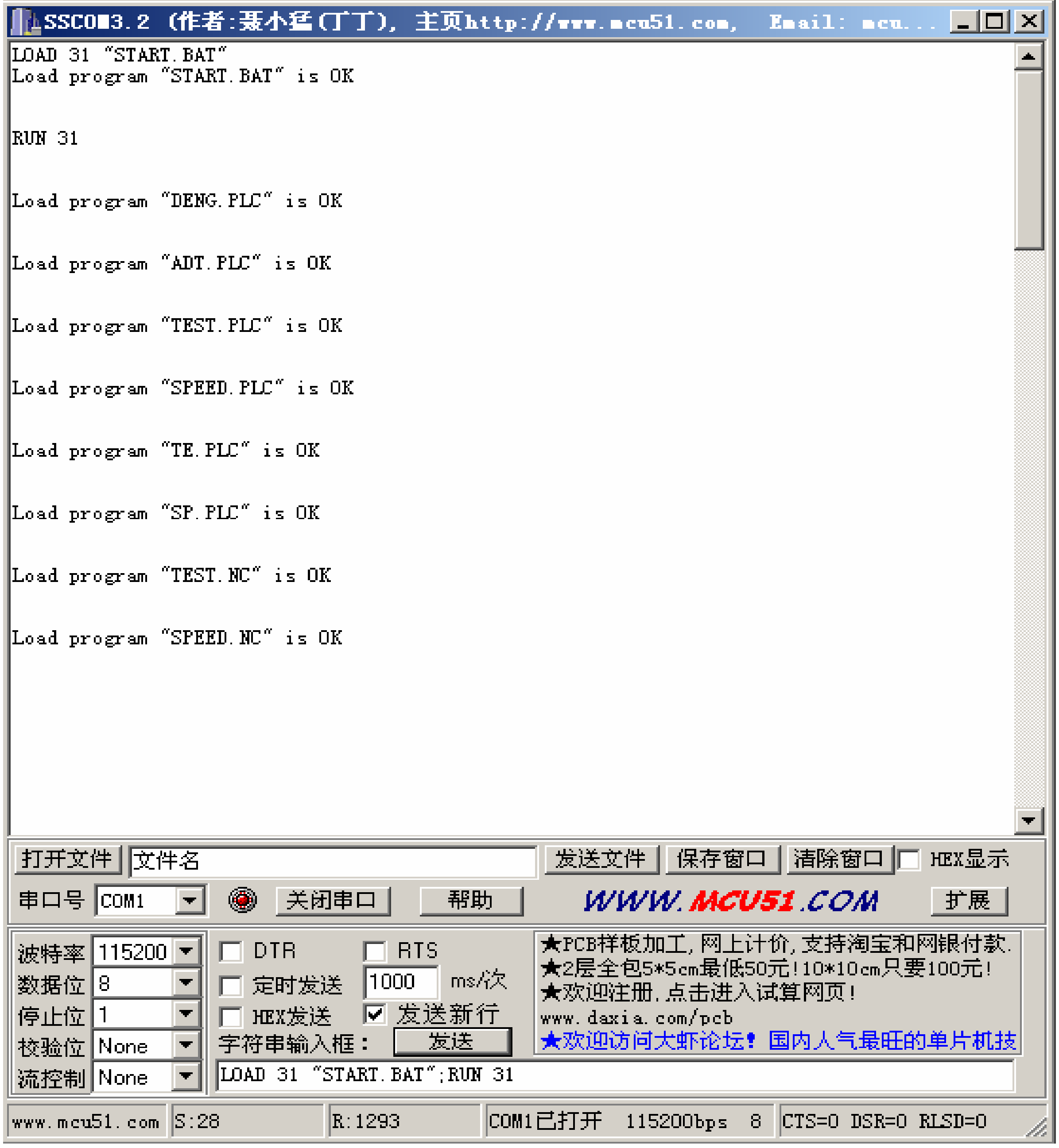
★PCB样板加工, 网上计价, 支持淘宝和网银付款.
★2层全包5*5cm最低50元!10*10cm只要100元!
★欢迎注册, 点击进入试算网页!
www.daxia.com/pcb
★欢迎访问大虾论坛! 国内人气最旺的单片机技

www.mcu51.com S:5 R:474 COM1已打开 115200bps 8 CTS=0 DSR=0 RLSD=0

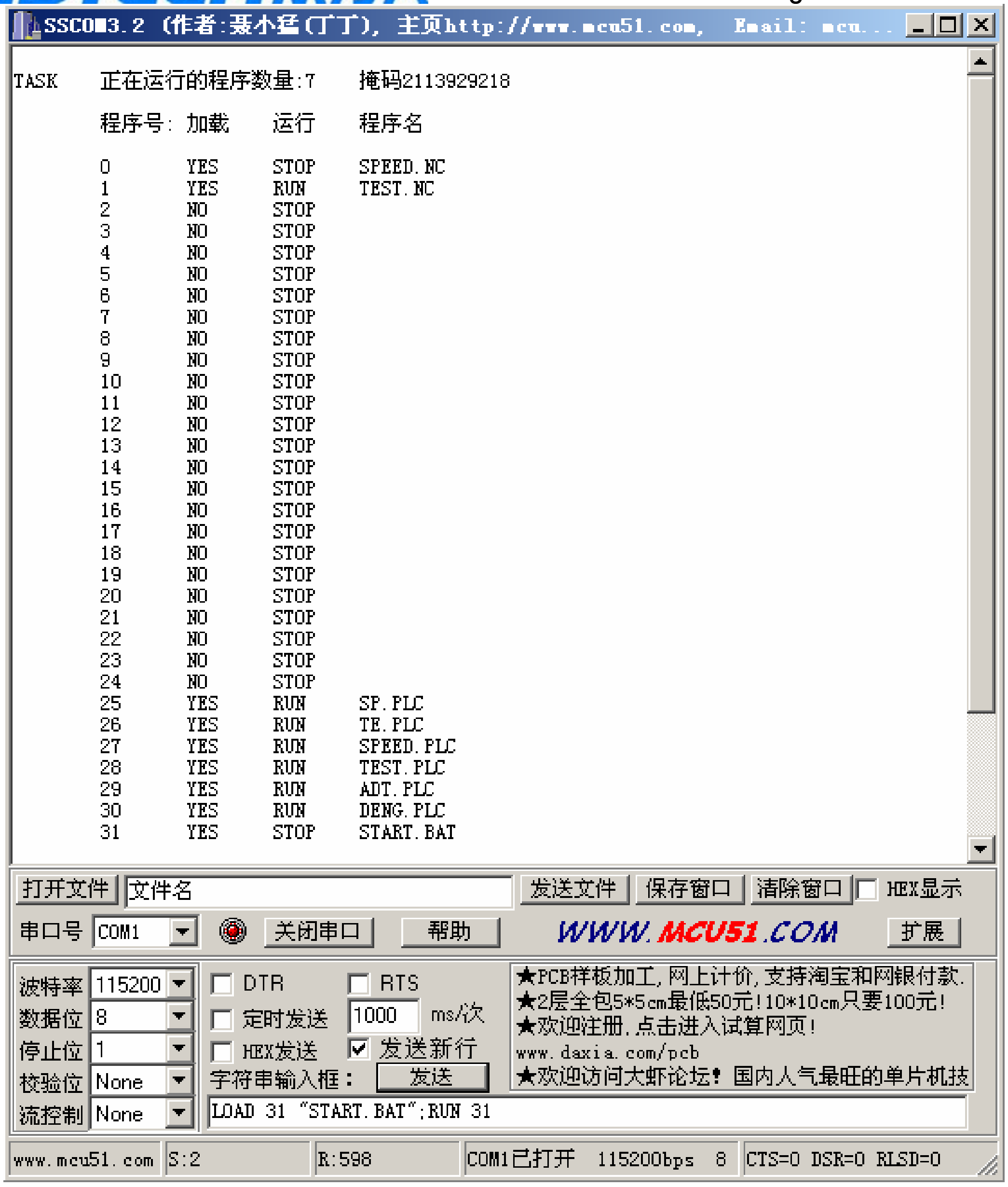
正在运行的程序数量：0 Running program quantity: 0   掩码 0 Mask 0
程序号：加载 Program No.: Load \ 运行 Run \ 程序名 Program name
打开文件/文件名 Open file: File name
串口号 Serial port No. \关闭窗口 Close serial port \ 帮助 Help
发送文件 Send file \ 保存窗口 Save window \ 清除窗口 Clear window \HEX 显示 HEX display
扩展 Extend
波特率 Baud rate / 数据位 Data bit / 停止位 Stop bit / 检验位 Parity / 流控位 Flow control
定时发送 Schedule send / HEX 发送 HEX send / 字符串输入框 String entry box
发送新行 Send new line
发送 Send

At this moment, the system isn't running any user program, and then we will execute LOAD 31 "START.BAT"; RUN 31;

This command will load and run a user program named "START.BAT", the No. of which is 31. This program is actually text file program written in C similar language style script supported by PMC controller. The function of this program is to load and run other programs in batch written by the user. Now, we will execute this command and then use the TASK command to view system operating status.


```
LOAD 31 "START.BAT"
Load program "START.BAT" is OK

RUN 31

Load program "DENG.PLC" is OK

Load program "ADT.PLC" is OK

Load program "TEST.PLC" is OK

Load program "SPEED.PLC" is OK

Load program "TE.PLC" is OK

Load program "SP.PLC" is OK

Load program "TEST.NC" is OK

Load program "SPEED.NC" is OK
```

打开文件/文件名 Open file: File name
串口号 Serial port No. \关闭窗口 Close serial port \ 帮助 Help
发送文件 Send file \ 保存窗口 Save window \ 清除窗口 Clear window \HEX 显示 HEX display
扩展 Extend
波特率 Baud rate / 数据位 Data bit / 停止位 Stop bit / 检验位 Parity / 流控位 Flow control
定时发送 Schedule send / HEX 发送 HEX send / 字符串输入框 String entry box
发送新行 Send new line
发送 Send



SSCOM3.2 (作者:聂小猛(丁丁), 主页http://www.mcu51.com, Email: mcu...

```
TASK     正在运行的程序数量:7      掩码2113929218

         程序号: 加载     运行     程序名

         0        YES      STOP     SPEED.NC
         1        YES      RUN      TEST.NC
         2        NO       STOP
         3        NO       STOP
         4        NO       STOP
         5        NO       STOP
         6        NO       STOP
         7        NO       STOP
         8        NO       STOP
         9        NO       STOP
         10       NO       STOP
         11       NO       STOP
         12       NO       STOP
         13       NO       STOP
         14       NO       STOP
         15       NO       STOP
         16       NO       STOP
         17       NO       STOP
         18       NO       STOP
         19       NO       STOP
         20       NO       STOP
         21       NO       STOP
         22       NO       STOP
         23       NO       STOP
         24       NO       STOP
         25       YES      RUN      SP.PLC
         26       YES      RUN      TE.PLC
         27       YES      RUN      SPEED.PLC
         28       YES      RUN      TEST.PLC
         29       YES      RUN      ADT.PLC
         30       YES      RUN      DENG.PLC
         31       YES      STOP     START.BAT
```

打开文件 | 文件名 | 发送文件 | 保存窗口 | 清除窗口 | HEX显示

串口号 COM1 | 关闭串口 | 帮助 | WWW.MCU51.COM | 扩展

波特率 115200
数据位 8
停止位 1
校验位 None
流控制 None

DTR　RTS
定时发送 1000 ms/次
HEX发送　[✓ 发送新行]
字符串输入框: 发送

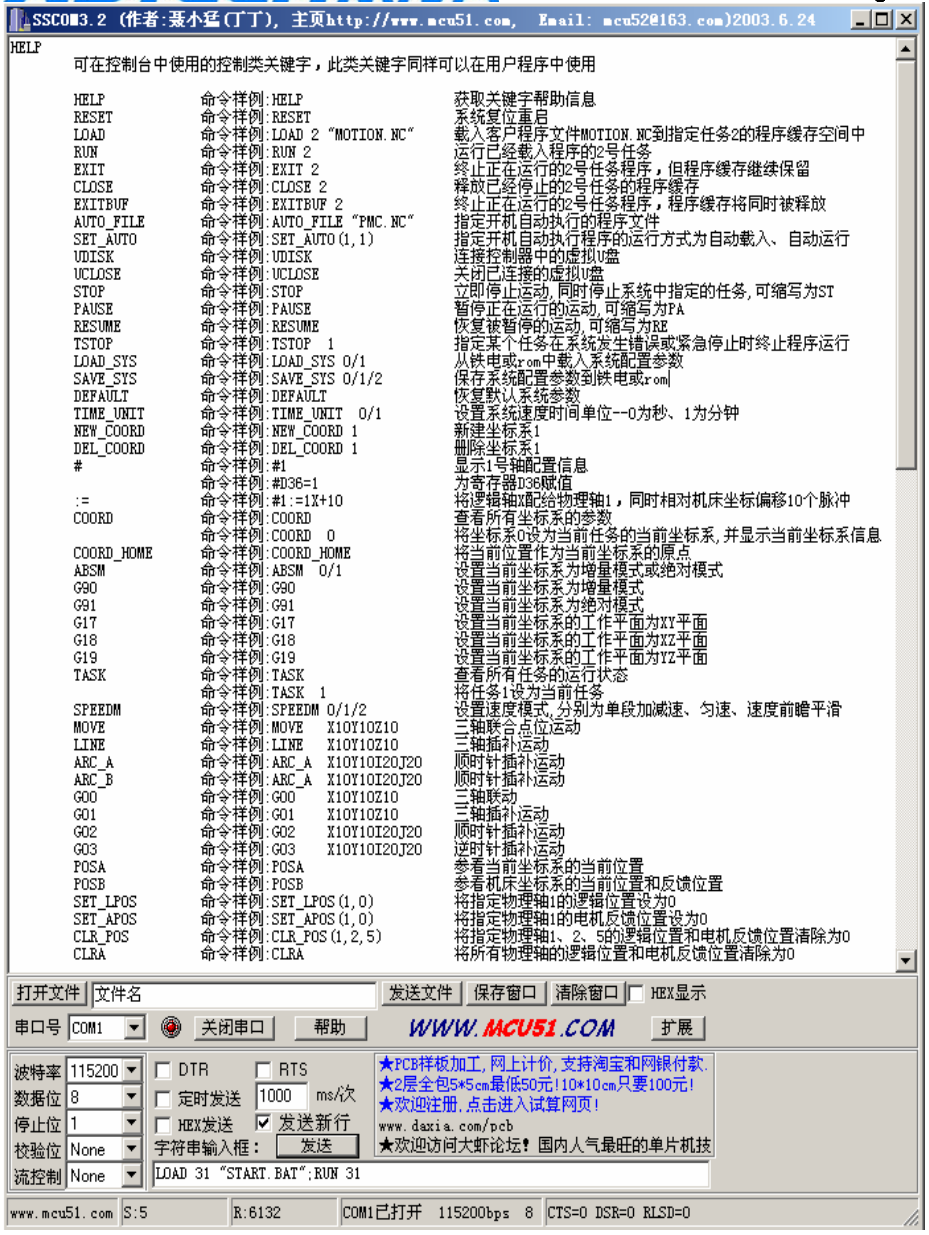
★PCB样板加工,网上计价,支持淘宝和网银付款.
★2层全包5*5cm最低50元!10*10cm只要100元!
★欢迎注册,点击进入试算网页!
www.daxia.com/pcb
★欢迎访问大虾论坛! 国内人气最旺的单片机技

LOAD 31 "START.BAT";RUN 31

www.mcu51.com | S:2 | R:598 | COM1已打开 115200bps 8 | CTS=0 DSR=0 RLSD=0

正在运行的程序数量：7 Running program quantity: 7　掩码 2113929218 Mask 2113929218
程序号：加载 Program No.: Load \ 运行 Run \ 程序名 Program name
打开文件/文件名 Open file: File name
串口号 Serial port No. \关闭窗口 Close serial port \ 帮助 Help
发送文件 Send file \ 保存窗口 Save window \ 清除窗口 Clear window \HEX 显示 HEX display
扩展 Extend
波特率 Baud rate / 数据位 Data bit / 停止位 Stop bit / 检验位 Parity / 流控位 Flow control
定时发送 Schedule send / HEX 发送 HEX send / 字符串输入框 String entry box
发送新行 Send new line
发送 Send

PMC console supports commands with various functions. If the user doesn't know the function and writing of instructions, please use HELP command to get the information of all commands and keywords in the console, as shown below:



```
SSCOM3.2 (作者:聂小猛(丁丁), 主页http://www.mcu51.com,  Email: mcu52@163.com)2003.6.24
HELP
        可在控制台中使用的控制类关键字，此类关键字同样可以在用户程序中使用

        HELP            命令样例:HELP                   获取关键字帮助信息
        RESET           命令样例:RESET                  系统复位重启
        LOAD            命令样例:LOAD 2 "MOTION.NC"     载入客户程序文件MOTION.NC到指定任务2的程序缓存空间中
        RUN             命令样例:RUN 2                  运行已经载入程序的2号任务
        EXIT            命令样例:EXIT 2                 终止正在运行的2号任务程序，但程序缓存继续保留
        CLOSE           命令样例:CLOSE 2                释放已经停止的2号任务的程序缓存
        EXITBUF         命令样例:EXITBUF 2              终止正在运行的2号任务程序，程序缓存将同时被释放
        AUTO_FILE       命令样例:AUTO_FILE "PMC.NC"     指定开机自动执行的程序文件
        SET_AUTO        命令样例:SET_AUTO(1,1)          指定开机自动执行程序的运行方式为自动载入、自动运行
        UDISK           命令样例:UDISK                  连接控制器中的虚拟U盘
        UCLOSE          命令样例:UCLOSE                 关闭已连接的虚拟U盘
        STOP            命令样例:STOP                   立即停止运动,同时停止系统中指定的任务,可缩写为ST
        PAUSE           命令样例:PAUSE                  暂停正在运行的运动,可缩写为PA
        RESUME          命令样例:RESUME                 恢复被暂停的运动,可缩写为RE
        TSTOP           命令样例:TSTOP  1               指定某个任务在系统发生错误或紧急停止时终止程序运行
        LOAD_SYS        命令样例:LOAD_SYS 0/1           从铁电或rom中载入系统配置参数
        SAVE_SYS        命令样例:SAVE_SYS 0/1/2         保存系统配置参数到铁电或rom
        DEFAULT         命令样例:DEFAULT                恢复默认系统参数
        TIME_UNIT       命令样例:TIME_UNIT  0/1         设置系统速度时间单位--0为秒、1为分钟
        NEW_COORD       命令样例:NEW_COORD 1            新建坐标系1
        DEL_COORD       命令样例:DEL_COORD 1            删除坐标系1
        #               命令样例:#1                     显示1号轴配置信息
                        命令样例:#D36=1                 为寄存器D36赋值
        :=              命令样例:#1:=1X+10              将逻辑轴X配给物理轴1，同时相对机床坐标偏移10个脉冲
        COORD           命令样例:COORD                  查看所有坐标系的参数
                        命令样例:COORD  0               将坐标系0设为当前任务的当前坐标系,并显示当前坐标系信息
        COORD_HOME      命令样例:COORD_HOME             将当前位置作为当前坐标系的原点
        ABSM            命令样例:ABSM  0/1              设置当前坐标系为增量模式或绝对模式
        G90             命令样例:G90                    设置当前坐标系为增量模式
        G91             命令样例:G91                    设置当前坐标系为绝对模式
        G17             命令样例:G17                    设置当前坐标系的工作平面为XY平面
        G18             命令样例:G18                    设置当前坐标系的工作平面为XZ平面
        G19             命令样例:G19                    设置当前坐标系的工作平面为YZ平面
        TASK            命令样例:TASK                   查看所有任务的运行状态
                        命令样例:TASK  1                将任务1设为当前任务
        SPEEDM          命令样例:SPEEDM 0/1/2           设置速度模式,分别为单段加减速、匀速、速度前瞻平滑
        MOVE            命令样例:MOVE   X10Y10Z10       三轴联合点位运动
        LINE            命令样例:LINE   X10Y10Z10       三轴插补运动
        ARC_A           命令样例:ARC_A  X10Y10I20J20    顺时针插补运动
        ARC_B           命令样例:ARC_A  X10Y10I20J20    顺时针插补运动
        G00             命令样例:G00    X10Y10Z10       三轴联动
        G01             命令样例:G01    X10Y10Z10       三轴插补运动
        G02             命令样例:G02    X10Y10I20J20    顺时针插补运动
        G03             命令样例:G03    X10Y10I20J20    逆时针插补运动
        POSA            命令样例:POSA                   参看当前坐标系的当前位置
        POSB            命令样例:POSB                   参看机床坐标系的当前位置和反馈位置
        SET_LPOS        命令样例:SET_LPOS(1,0)          将指定物理轴1的逻辑位置设为0
        SET_APOS        命令样例:SET_APOS(1,0)          将指定物理轴1的电机反馈位置设为0
        CLR_POS         命令样例:CLR_POS(1,2,5)         将指定物理轴1、2、5的逻辑位置和电机反馈位置清除为0
        CLRA            命令样例:CLRA                   将所有物理轴的逻辑位置和电机反馈位置清除为0
```

波特率	115200
数据位	8
停止位	1
校验位	None
流控制	None

串口号 COM1

☐ DTR ☐ RTS ☐ 定时发送 1000 ms/次 ☐ HEX发送 ☑ 发送新行 ☐ HEX显示

字符串输入框：

```
LOAD 31 "START.BAT";RUN 31
```

★PCB样板加工,网上计价,支持淘宝和网银付款.
★2层全包5*5cm最低50元!10*10cm只要100元!
★欢迎注册,点击进入试算网页!
www.daxia.com/pcb
★欢迎访问大虾论坛! 国内人气最旺的单片机技

www.mcu51.com S:5 R:6132 COM1已打开 115200bps 8 CTS=0 DSR=0 RLSD=0

可在控制台中使用的控制类关键字，此类关键字同样可以在用户程序中使用
Control keywords allowed in the console; these keywords also can be used in user program
命令样例 Command example
获取关键字帮助信息 Get help info on keywords
系统复位重启 Reset and restart the system
载入客户程序文件 MOTION.NC 到指定任务 2 的程序缓存空间中
Load client program file MOTION.NC into the program cache of specified task 2
运行已经载入程序的 2 号任务 Run the #2 task loaded in the program
终止正在运行的 2 号任务程序，但程序缓存继续保留 Stop running #2 task program, but the program cache is retained
释放已经停止的 2 号任务 Release the program cache of stopped #2 task
终止正在运行的 2 号任务程序，程序缓存将同时被释放 Stop running #2 task program, and the program cache is also released
指定开机自动执行的程序文件 Specify the startup program file
指定开机自动执行程序的运行方式为自动载入、自动运行
Specify the running mode of the startup program as auto-load or auto-run
连接控制器中的虚拟 U 盘 Connect to the virtual USB disk in the controller
关闭已连接的虚拟 U 盘 Close connected virtual USB disk
立即停止运动，同时停止系统中指定的任务，可缩写为 ST
Stop motion immediately, and also stop the specified task in the system, can abbreviate to ST
暂停正在运行的运动，可缩写为 PAPause running motion, can abbreviate to PA
恢复被暂停的运动，可缩写为 REResume the motion, can abbreviate to RE
指定某个任务在系统发生错误或紧急停止时终止程序运行
Specify the task to stop program running when the system has error or stops in emergency

从铁电或 rom 中载入系统配置参数 Load system configuration parameters from ferroelectric RAM or ROM
保存系统配置参数到铁电或 romSave system configuration parameters in ferroelectric RAM or ROM
恢复默认系统参数 Restore default system parameters
设置系统速度时间单位—0 为秒、1 为分钟 Set system speed time unit: 0: second, 1: minute
新建坐标系 1 Create new coordinate system 1
删除坐标系 1 Delete coordinate system 1
显示 1 号轴配置信息 Show the configuration info of #1 axis
为寄存器 D36 赋值 Evaluate register D36
将逻辑轴 X 配给物理轴 1，同时相对机床坐标偏移 10 个脉冲
Assign logic axis X to physical axis 1, and the relative machine tool coordinate system offsets 10 pulses
查看所有坐标系的参数 View the parameters of all coordinate systems
将坐标系 0 设为当前任务的当前坐标系，并显示当前坐标系信息
Set coordinate system 0 as the current coordinate system of current task, and show the info of current coordinate system
将当前位置作为当前坐标系的原点 Set current position as the home of current coordinate system
设置当前坐标系为增量模式或绝对模式 Set current coordinate system to increment mode or absolute mode
设置当前坐标系为增量模式 Set current coordinate system to increment mode
设置当前坐标系为绝对模式 Set current coordinate system to absolute mode
设置当前坐标系的工作平面为 XY 平面 Set working plane of current coordinate system as XY plane
设置当前坐标系的工作平面为 XZ 平面 Set working plane of current coordinate system as XZ plane
设置当前坐标系的工作平面为 YZ 平面 Set working plane of current coordinate system as YZ plane
查看所有任务的运行状态 View the running status of all tasks
将任务 1 设为当前任务 Set task 1 as current task
设置速度模式，分别为单段加减速、匀速、速度前瞻平滑
Set the speed mode, including single section acceleration/deceleration, constant, look-ahead, smoothing
三轴联合点位运动 Three axes joint point motion
三轴插补运动 Three axes interpolation motion
顺时针插补运动 Clockwise interpolation motion
顺时针插补运动 Clockwise interpolation motion
三轴联动 Three axes linkage
三轴插补运动 Three axes interpolation motion
顺时针插补运动 Clockwise interpolation motion
逆时针插补运动 Counterclockwise interpolation motion
参看当前坐标系的当前位置 View current position of current coordinate system
参看机床坐标系的当前位置和反馈位置 View current position and feedback position of machine tool coordinate system
将指定物理轴 1 的逻辑位置设为 0 Set the logic position of specified physical axis 1 to 0
将指定物理轴 1 的电机反馈位置设为 0 Set the motor feedback position of specified physical axis 1 to 0
将指定物理轴 1、2、5 的逻辑位置和电机反馈位置清除为 0
Clear the logic position and motor feedback position of specified physical axis 1, 2, 5
将所有物理轴的逻辑位置和电机反馈位置清除为 0 Clear the logic position and motor feedback position of all physical axes

打开文件/文件名 Open file: File name
串口号 Serial port No. \关闭窗口 Close serial port \ 帮助 Help
发送文件 Send file \ 保存窗口 Save window \ 清除窗口 Clear window \HEX 显示 HEX display
扩展 Extend
波特率 Baud rate / 数据位 Data bit / 停止位 Stop bit / 检验位 Parity / 流控位 Flow control
定时发送 Schedule send / HEX 发送 HEX send / 字符串输入框 String entry box
发送新行 Send new line
发送 Send

All console commands and detailed function of program keywords are described in Instruction Programming.

7. Instruction Programming

7.1. Programming basics

In the programming instruction system of this PMC, all instructions and keywords are in capitalized letters, the keywords are separated with space or tab, the instructions are ended with ';' or newline symbol, character strings are contained with "", and program files are named in 8+3 short file name. Use {...} as the block boundary, and support using () to nest expressions. If the program has non-defined instructions, the program will report error automatically when executing.

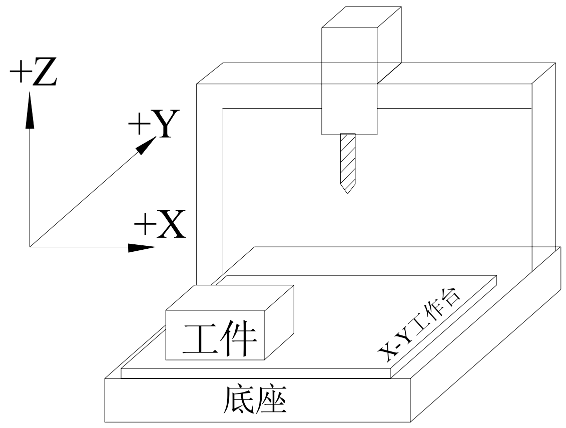
7.1.1 Keyword and instructions table

Keyword or instruction	Mode	Console	Instruction and expression example	Function
HELP		Yes	HELP	Get help info on keywords
RESET		Yes	RESET	System software resets and restarts
LOAD		Yes	LOAD 31 "START.BAT"	Load specified program into task 31
TASK		Yes	TASK	View the status of all tasks of the system
RUN		Yes	RUN 31	Run specified task 31
EXIT		Yes	EXIT 31	End running specified task 31
CLOSE		Yes	CLOSE 31	Close the program cache of specified task 31
AUTO_FILE		Yes	AUTO_FILE "START.BAT"	Specify program START.BAT as startup program
SET_AUTO		Yes	SET_AUTO(1, 1)	Set the start mode of startup programs, auto load or auto run
UDISK		Yes	UDISK	Connect analog USB disk
UCLOSE		Yes	UCLOSE	Disconnect analog USB disk
STOP		Yes	STOP	Stop motion, and the motion program
PAUSE		Yes	PAUSE	Pause
RESUME		Yes	RESUME	Resume
LOAD_SYS		Yes	LOAD_SYS 1/2/5/6	Load system parameters from ferroelectric RAM or ROM
SAVE_SYS		Yes	SAVE_SYS 1/2/3/5/6/7	Save system parameters into ferroelectric RAM or ROM
DEFAULT		Yes	DEFAULT	Restore default system parameters
TIME_UNIT		Yes	TIME_UNIT 0/1	Set the time unit of system speed to sec or min
NEW_COORD		Yes	NEW_COORD 1	Create new coordinate system 1
DEL_COORD		Yes	DEL_COORD 1	Delete coordinate system 1
COORD		Yes	COORD	View the configuration info of all coordinate systems
		Yes	COORD 1	Set workpiece coordinate system 1 as current coordinate system
COORD_HOME	Yes	Yes	COORD_HOME	Set current position as the home of current coordinate system
ABSM	Yes	Yes	ABSM 0/1	Set current coordinate system in increment or absolute coordinate mode
G90	Yes	Yes	G90	Set current coordinate system in absolute coordinate mode
G91	Yes	Yes	G91	Set current coordinate system in increment coordinate mode
G17	Yes	Yes	G17	Set current as XY motion plane
G18	Yes	Yes	G18	Set current as XZ motion plane
G19	Yes	Yes	G19	Set current as YZ motion plane
#		Yes	#1	Show the configuration info of #1 physical axis
		Yes	#D36	Access D36 register
:=	Yes	Yes	#1:=X+20	Assign X axis to #1 physical axis, and offset 20 basic unit sizes
SET_IP		Yes	SET_IP "192.168.0.123"	Set PMC IP address, save system parameters and

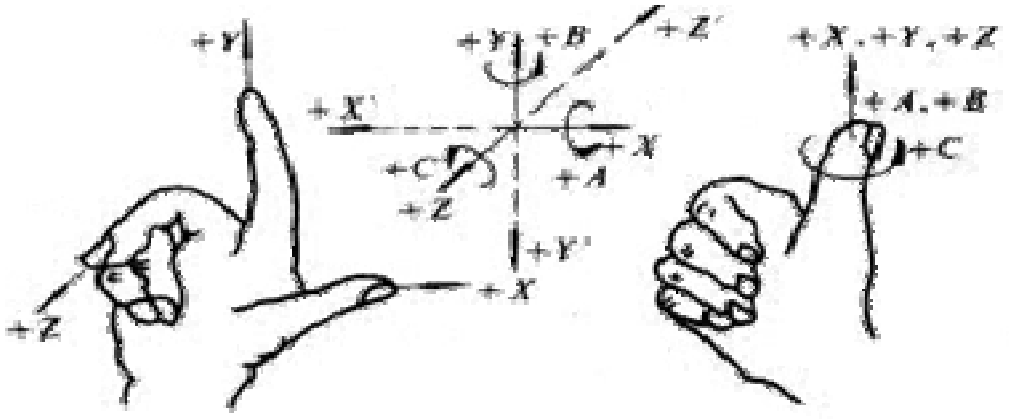
				restart to take effect
SET_MAC		Yes	SET_MAC "11-22-33-44-55-66"	Set PMC MAC address, save system parameters and restart to take effect
SET_UART		Yes	SET_UART (0,115200,8,0,1)	Set configuration parameters for serial port 0
SET_DATE		Yes	SET_DATA(2011,03,2,22)	Set system date to Tuesday, April 22nd, 2011
SET_TIME		Yes	SET_TIME (15,31,50)	Set system time to 15:31:50
DATE		Yes	DATE	Show system date
TIME		Yes	TIME	Show system time
IPCONFIG		Yes	IPCONFIG	Show system IP/MAC address and serial port configuration info
VERSION		Yes	VERSION	Show system software version
IN		Yes	IN(0)	Read status of input port 0
RIN		Yes	RIN(0)	Read cache status of input port 0
WIN		Yes	WIN(100,1)	Write 1 to virtual input port 100
OUT		Yes	OUT(0,1)	Write 1 to output port 0
ROUT		Yes	ROUT(0)	Read cache status of output port 0
MOVE/G00	Yes	Yes	MOVE X100Y100	X axis and Y axis moves to specified coordinates in quick positioning mode; the positions of axes aren't related, and the motion track may be linear or curved broken line
			G00 X100Y100	
LINE/G01	Yes	Yes	LINE X100Y100	X axis and Y axis moves to specified coordinates in linear interpolation mode, the interpolation speed can be specified to 3000mm/min with F
			G01 X100Y100F3000	
ARC_A/G02	Yes	Yes	ARC_A X40Y40I20J20	CW arc interpolation, arc start coordinate is current point, end coordinate is X40Y40, the coordinates of circle center offset 20 in both X axis and Y axis direction
			G02 X40Y40I20J20	
			ARC_A X40Y40R20	CW arc interpolation, arc start coordinate is current point, end coordinate is X40Y40, arc radius is 20, arc angle < π
			G02 X40Y40R20	
ARC_B/G03	Yes	Yes	ARC_B X40Y40I20J20	CCW arc interpolation, arc start coordinate is current point, end coordinate is X40Y40, the coordinates of circle center offset 20 in both X axis and Y axis direction
			G03 X40Y40I20J20F3000	
			ARC_B X40Y40R20	CCW arc interpolation, arc start coordinate i current point, end coordinate is X40Y40, ar radius is 20
			G03 X40Y40R20	
F	Yes	Yes	G01 X10Y10F3600	Set the feed speed of linear interpolation as unit length/unit time
POSA		Yes	POSA	Check the position of current coordinate system
POSB		Yes	POSB	Check the position of machine tool coordinate system
SET_LPOS		Yes	SET_LPOS(1,1000)	Set the pulse logic position of physical axis 1
SET_APOS		Yes	SET_APOS(1,1000)	Set the encoder position of physical axis 1
CLRA		Yes	CLRA	Clear the logic and actual positions of all physical axes
SOFT_LIMIT		Yes	SOFT_LIMIT(2,1,100000, 100000)	Set the software limit mode of physical axis 2
SET_STOPA		Yes	SET_STOPA(2,1,0)	Set hardware STOP0 signal of physical axis 2 effective, trigger voltage level to low
SET_STOPB		Yes	SET_STOPB(2,1,0)	Set hardware STOP1 signal of physical axis 2 effective, trigger voltage level to low
HOME		Yes	HOME(1,0,1)	Physical axis 1 and 3 home
SIN		Yes	SIN(30)	Evaluate sine function, input unit is circumferential angle
COS		Yes	COS(30)	Evaluate cosine function, input unit is circumferential angle

TAN		Yes	TAN(30)	Evaluate tangent function, input unit is circumferential angle
ASIN		Yes	ASIN(0.5)	Evaluate arcsine function, return unit is circumferential angle
ACOS		Yes	ACOS(0.5)	Evaluate arc cosine function, return unit is circumferential angle
ATAN		Yes	ATAN(0.5)	Evaluate arc tangent function, input unit is circumferential angle
SQRT		Yes	SQRT(2)	Evaluate square root function
EXP		Yes	EXP(2)	Evaluate the exponential function with natural number e as the base
LN		Yes	LN(2)	Evaluate the logarithmic function with natural number e as the base
LOG		Yes	LOG(2)	Evaluate the logarithmic function with 10 as the base
ABS		Yes	ABS(-123)	Evaluate absolute value function
&&		Yes	1&&2	Logical AND operator
||		Yes	1||2	Logical OR operator
^^		Yes	1^^2	Logical XOR operator
!		Yes	!2	Logical NOT operator
&		Yes	1&2	Bitwise AND operator
|		Yes	1|2	Bitwise OR operator
^		Yes	1^2	Bitwise XOR operator
~		Yes	~1	Bitwise negation operator
+		Yes	1+2	Addition arithmetic operator
			+	Positive value symbol
-		Yes	3-2	Subtraction arithmetic operator
			-2	Negative value symbol
*		Yes	2*3	Multiplication arithmetic operator
/		Yes	3/2	Division arithmetic operator
%		Yes	3%2	Remainder operator
IF		No	IF(IN(0)) {…}	Check whether the expression condition is true
ELSE		No	ELSE {…}	If the IF expression isn't true, execute the sentence after ELSE
ELSE IF		No	ELSE IF {…}	ELSE…IF condition nesting
WHILE		No	WHILE(1) {…}	Cycling condition determination
DO-WHILE		No	DO{…}WHILE(…)	DO-WHILE loop structure
BREAK		No	BREAK	Out of the WHILE loop
GOTO		No	GOTO N1111	Jump to N1111 program label unconditionally
CALL		No	CALL O123	Call subroutine O123
RETURN		No	RETURN	Return from subroutine
;		Yes	;	Instruction end symbol; the function is same to Enter; generally used to differentiate multiple instructions in same line of code
(…)		Yes	(3*(2+1)-4)/3	Expression boundary symbol
{…}		No	{…}	Block boundary symbol
//		No	//……	Program note symbol, ignoring the later content of the program

7.1.2 Motion direction and naming of control axis



工件 Workpiece　　X-Y 工作台 X-Y workbench　　底座 Base



This system can control the quick motion of random six axes, feed interpolation of random one, two or three axes, and first four, five or six axes interpolation.

Axis direction definition uses Cartesian coordinate system, as follows (facing to the machine tool):

Z: The upward/downward motion of the tool relative to the workpiece is Z axis motion; the upward motion is Z axis positive motion, and the downward motion is Z axis negative motion.

X: The leftward/rightward motion of the tool relative to the workpiece is X axis motion; the leftward motion is X axis negative motion, and the rightward motion is X axis positive motion.

Y: The frontward/backward motion of the tool relative to the workpiece is Y axis motion; the frontward motion is Y axis positive motion, and the backward motion is Y axis negative motion.

A,B,C: They are usually used as rotation coordinate axes, as well as other auxiliary motion. As rotation coordinate axis, the positive direction is the positive direction of axis X, Y, Z, and is determined according to the forward direction of right hand thread.

Note: The motion of axis X, Y, Z, A, B, C in this Manual is always the motion of tool relative to the workpiece, i.e. suppose that the workpiece coordinate system has been set.

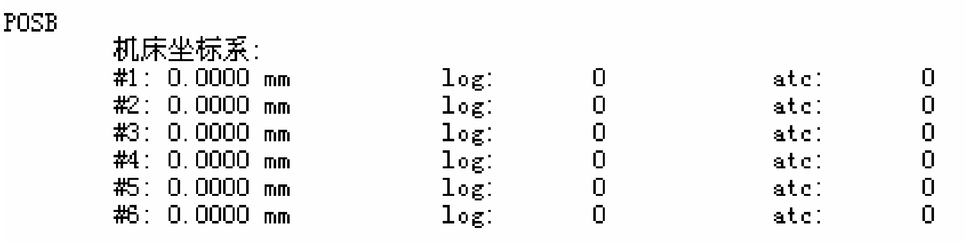
7.1.3 Coordinate systems of machine tool and workpiece

1) Machine tool coordinate system

The machine tool coordinate system is the fixed coordinate system of the machine tool. The coordinate system is established through the operation of returning to reference point when NC is electrified every time. The default mapping relationship between logic axes and physical axes of the machine tool coordinate system is #1:=X, #2:=Y, #3:=Z, #4:=A, #5:=B, #6:=C; COORD 0 instruction is used to select machine tool coordinate system.

View machine tool coordinate system position info instruction
Format: POSB
No parameter

This instruction makes console show current position info of the machine tool coordinate system, as shown below:



```
POSB
        机床坐标系:
        #1: 0.0000 mm        log:      0         atc:      0
        #2: 0.0000 mm        log:      0         atc:      0
        #3: 0.0000 mm        log:      0         atc:      0
        #4: 0.0000 mm        log:      0         atc:      0
        #5: 0.0000 mm        log:      0         atc:      0
        #6: 0.0000 mm        log:      0         atc:      0
```

机床坐标系 Machine tool coordinate system:

Set machine tool coordinate system logic position instruction
Format: SET_LPOS(x,n)

The first parameter x is specified physical axis No., the range of which is 1-6
The second parameter n is the logic position, and the unit is pulse.
The function of this instruction is to change the current position of the machine tool coordinate system. This operation will change the machine tool home and is seldom used in actual work. It is usually the auxiliary function for program testing.

Set machine tool coordinate system encoder position instruction
Format: SET_APOS(x,n)

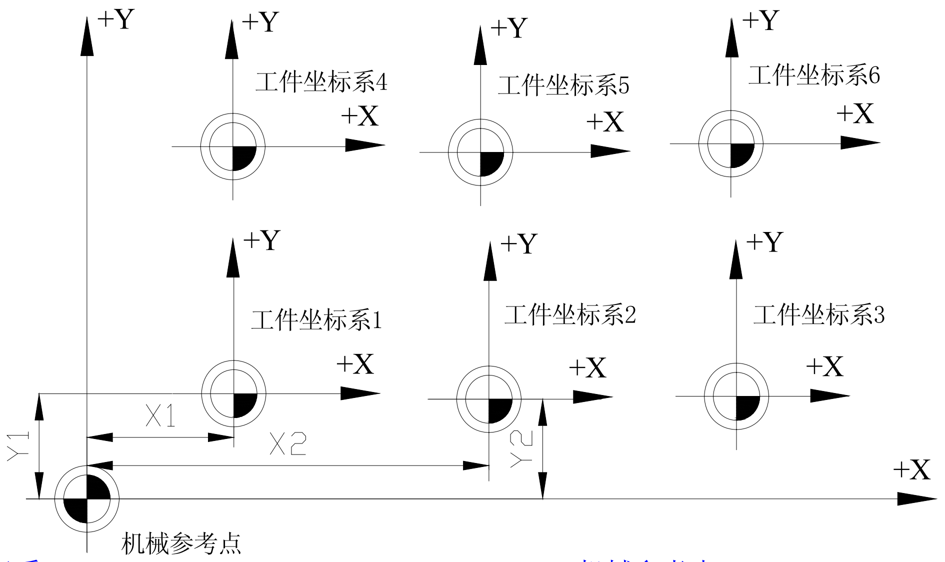
The first parameter x is specified physical axis No., the range of which is 1-6
The second parameter n is encoder feedback position
The function of this instruction is to change the current position of the machine tool coordinate system encoder.

Set machine tool coordinate system home instruction
Format: CLRA
No parameter

This instruction will set current point as the home of the machine tool coordinate system, and the logic position and encoder position will be cleared. This instruction is seldom used in actual work and is usually the auxiliary function for program testing.

2) Workpiece coordinate system

Workpiece coordinate system is used for program processing, and will set certain datum point on the workpiece as the coordinate system of the origin. Generally, the programmer doesn't know the position of the processed workpiece on the machine tool when starts programming, and thus the workpiece program usually uses a point on the workpiece as the reference point to write the processing program. Therefore, the coordinate system created with this reference point is the workpiece coordinate system. When the processed workpiece is fixed on the machine tool workbench, first move the tool to specified workpiece reference point and set the machine tool coordinates of this point as the coordinate origin of the workpiece, and thus the tool will use this workpiece coordinate system as the reference system when the system is executing processing program and process according to program instruction. Therefore, the home offset function of coordinate system is very important for CNC machine tool. This system allows establishing and setting up to 32 workpiece coordinate systems, and workpiece coordinate system 0 is established by default when PMC system is electrified.



:工件坐标系 Workpiece coordinate system :机械参考点 Mechanical reference point

New coordinate system instruction

Format: NEW_COORD n

N: workpiece coordinate system code 0-31; new coordinates are consistent with machine tool coordinate system by default. The user can set the offset of every workpiece coordinate system relative to machine tool coordinate system and mapping of logic axes through axis configuration instruction.

Axis configuration instruction

Format: #n:=x+n

The first parameter n is specified physical axis, the range of which is 1-6

The second parameter x is specified logic axis, the range of which is X, Y, Z, A, B, C

The third parameter n indicates the offset of current axis in current workpiece coordinate system relative to machine tool coordinate system.

A typical example of coordinate system setting instruction is #1:=X+10, i.e. logic axis X is adapted to #1 physical axis, and this axis offsets +10 dimension units relative to machine tool coordinate system.

Select current coordinate system instruction

Format: COORD n

n is the specified workpiece coordinate system

Every task can use specified coordinate system as current coordinate system of the task through COORD n instruction. If this coordinate system isn’t established, the system reports error automatically. Current motion must be stopped before changing current coordinate system.

Delete coordinate system instruction

Format: DEL_COORD n

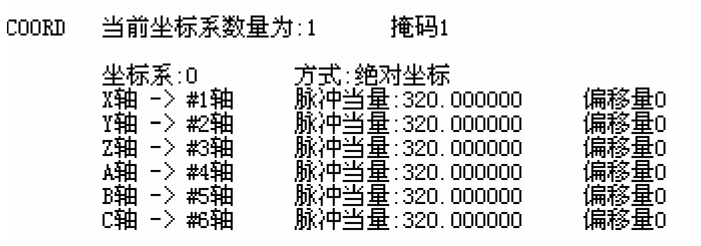
n is the specified workpiece coordinate system

The user can use DEL_COORD n instruction to delete a redundant workpiece coordinate system and trim related statistics info.

View system coordinates info instruction

Format: COORD no parameter

This command can be used to view the configuration info of all coordinate systems in the system in the console, as shown below



```
COORD    当前坐标系数量为:1        掩码1

         坐标系:0            方式:绝对坐标
         X轴 -> #1轴         脉冲当量:320.000000      偏移量0
         Y轴 -> #2轴         脉冲当量:320.000000      偏移量0
         Z轴 -> #3轴         脉冲当量:320.000000      偏移量0
         A轴 -> #4轴         脉冲当量:320.000000      偏移量0
         B轴 -> #5轴         脉冲当量:320.000000      偏移量0
         C轴 -> #6轴         脉冲当量:320.000000      偏移量0
```

Current coordinate system quantity: 1	Mask 1	
Coordinate system: 0	Mode: absolute coordinates	
X axis → # 1 axis	Pulse equivalent:	Offset
Y axis → # 2 axis	Pulse equivalent:	Offset
Z axis → # 3 axis	Pulse equivalent:	Offset
A axis → # 4 axis	Pulse equivalent:	Offset
B axis → # 5 axis	Pulse equivalent:	Offset
C axis → # 6 axis	Pulse equivalent:	Offset

Set workpiece coordinate system home instruction

Format: COORD_HOME no parameter

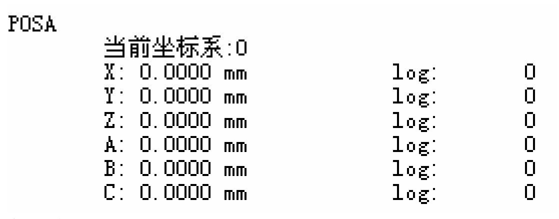
This command can be used to set current position as the home of current workpiece coordinate system; this instruction will re-calculate and set the offset of current coordinate system relative to machine tool coordinate system.

View current coordinate system position info instruction

Format: POSA

No parameter

This instruction will make the console show the current position info of current coordinate system, as shown below



```
POSA
         当前坐标系:0
         X:  0.0000 mm              log:      0
         Y:  0.0000 mm              log:      0
         Z:  0.0000 mm              log:      0
         A:  0.0000 mm              log:      0
         B:  0.0000 mm              log:      0
         C:  0.0000 mm              log:      0
```

当前坐标系：0 Current coordinate system: 0

3) Absolute coordinates and relative coordinates programming

The coordinates in motion instructions have two expression modes: absolute and increment. In absolute coordinate mode, the coordinates of the motion end point in current coordinate system are specified; in increment coordinate mode, the distance of every coordinate axis relative to starting point motion is specified.

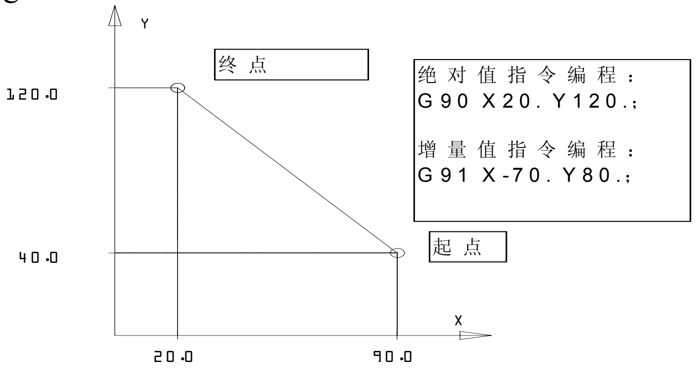
Three modes are available to set absolute coordinates and increment coordinates:

1) Use script instruction **ABSM 0/1**, 0- increment mode coordinates, 1- absolute mode coordinates

2) Use G code instruction, **G90:** absolute mode coordinates, **G91:** increment mode coordinates

3) By setting the value of register **D36**, 0- increment mode coordinates, 1- absolute mode coordinates

Example diagram



终点 Ending point 绝对值指令编程 Absolute instruction programming
增量值指令编程 Increment instruction programming 起点 Starting point

Through above example, we can understand the programming in absolute value mode and increment mode better

7.1.4 Mode and modeless instructions

The so-called modal function is always valid once a code is designated in the current block until another code of same group appears in the block, and this instruction doesn't need to be specified if it is used by next block. The so-called modeless function refers to a code is only valid in the block where it is in, and should be re-specified if the next block uses it.

For example:

G01 X150. Y25. F100; (linear interpolation to X150, Y25)

X50. Y75. F120; (linear interpolation to X50, Y75, G01 is mode instruction and can be omitted)

X0; (linear interpolation to X0, Y75. F120 is mode instruction and can be omitted)

The mode properties of all instructions can be viewed 7.1.1 Instruction Table.

7.1.5 Program structure and operation mechanism

PMC provides C similar language style program code structure, which is much different from standard C language. PMC only supports capitalized program keywords, doesn't have main function entry in the program, doesn't support the function libraries, variable statement, pointer or array operation in standard C, and only supports program cycles such as IF, ELSE, ELSE IF, WHILE, DO-WHILE, BREAK, GOTO, CALL, program structure function processed by branches and motion control related special functions, as well as mathematical and logic expressions; in addition, PMC also support commonly used G code motion function preparation word. The keywords are separated with space and tab. The program has illegal keywords during executing, the system will report error automatically and stop current motion task.

The instruction code of every line is called as a block, which has up to 64 ASIC characters (the note statement after // isn't limited). The starting position of the block has a block No. started with character N, followed by four digits value, e.g. N1234. This block No. must be started from the top line, and is used for GOTO addressing. The value of block No. and block position do not have relation. It can be any value, but can't be repeated. In this Manual, the instruction and function sentences are ended with ";" or newline symbol. Instruction or function sentences are basic elements of blocks. One block can't be constituted with several instruction sentences, and several blocks constitute a processing program.

The access of all internal registers of PMC must be identified with prefix #, e.g. #E100=123.
IF, ELSE, ELSE IF, WHILE, DO-WHILE can use { } to bracket several instruction sentences as a large conditional execution block, and allow multi-level condition nesting. All register values, numerical value, logic calculation and IO status can be used as the expressions of condition judgment.

1) Main program

The processing programs comprise main program and subroutine. Every processing file is an independent main program. Generally, the main programs are classified into startup programs (or batch programs), PLC programs, and motion control programs according to functions. Program types do not have strict boundary, and aren't distinguished according to file names.

Program loading instruction
Format: LOAD n "xxxxxxxx.xxx"
The first parameter n is the task No.
The second parameter is the character string in "", which is the file name of the instruction program written by the user.
Function: load the program files written by the user into the cache of specified task; if no specified program or other errors are found, PMC system will report error.

Program running instruction
Format: RUN n
Parameter n is the task No.
Function: run specified, loaded program task; if the program isn't loaded, PMC system will report error.

Exit program instruction
Format: EXIT n
Parameter n is the task No.
Function: stop specified running task, and the program cache is still valid when the task is stopped

Close program cache instruction
Format: CLOSE n
Parameter n is the task No.
Function: close specified, stopped program cache

Set startup program file instruction
Format: AUTO_FILE("xxxxxxxx.xxx")
Function: set specified program file as startup program; this program name is actually saved in C73-C76 registers, and will take effect after the system parameters are saved.

Set startup mode instruction
Format: SET_AUTO(n,n)
The first parameter sets whether load startup programs automatically after started, the value of which is 0 or 1, and 1 is valid.
The second parameter sets whether run startup programs automatically after started, the value of which is 0 or 1, and 1 is valid.
Function: set the start mode of startup files; the two parameters are saved in C71 and C72 registers respectively.

When PMC is electrified, the system will check C71 and C72 register parameters automatically. If the two parameter values are valid, load and execute the startup program files specified in C73-C76. The function of this program is to load or run other applications written by the user, and make the system establish working state. C71 and C72 parameters of PMC are 1 by default. The system loads and runs default startup program "START.BAT" automatically, the program No. is 31, and the priority is the lowest. The user can modify startup mode through command console, and use SAVE_SYS 1/2/5/6/7 to save system parameters.

A typical demonstration code of the startup program follows:

```
//"START.BAT" startup program file, used to load and run user programs in batch
IF(!(#C65 & (1<<30)))  //check whether program 30 has been loaded
{
   LOAD 30 "TEST.PLC"   //load program TEST.PLC into the cache of program 30
   RUN 30               //run program 30
}
IF(!(#C65 & (1<<1)))    // check whether program 1 has been loaded
{
   LOAD 1 "TEST.NC"// load program TEST.NC into the cache of program 1
   RUN 1                //run program 1
}
EXIT                    //exit startup program
```

Generally, the startup program runs only once and exits automatically, but the program cache will be always valid. If the user wants to destruct useless program cache, it can be realized in console or other programs with **CLOSE 31**; before destructing a program cache, this program should be stopped first.
PLC program usually achieves IO motion control through scanning the state of IO or register variables. A typical demonstration code of the PLC program follows:

```
    //Flash LED demonstration program, 8 output IO output high voltage level in turn, at
200ms interval
    #E100=0                // initialization parameter
    #E101=0
    WHILE(1)               // the program has been running and never exits
    {
    IF(IN(10))         //if input point 10 is in high voltage level, execute flash LED
    {
        IF(#E100==0)       //check variable value of #E100 to initialize state of flash
LED
        {
            OUT(0,0)
            OUT(1,0)
            OUT(2,0)
            OUT(3,0)
            OUT(4,0)
            OUT(5,0)
            OUT(6,0)
            OUT(7,0)
```

```
                #E100=1             // flash LED has initialized the label
                 #E101=0            // flash LED S/N
            }
            OUT(#E101,0)            // clear last flash LED state
            IF(#E101>=7)            // check whether flash LED arrives at boundary
                #E101=0
            ELSE
                 #E101=#E101+1          // prepare this action for the flash LED
            OUT(#E101,1)            // this flash LED outputs high voltage level
        }ELSE
        {
            #E100=0
            #E101=0
        }
        DELAY(200)                      //delay 200ms
    }
```

Task view instruction

Format: TASK

No parameter

Function: view the running state of every task in PMC system, including program loading and program running, mapping relationship between every task and program file name. C62 register saves the total running tasks of PMC, C63 register saves the total running task mask parameters of PMC, #C64 register saves total programs loaded in program cache of PMC, and C65 register saves the program masks loaded in program cache of PMC.

2) Subroutine

PMC subroutines are the modular function codes written in main programs. Subroutine name starts with character O, followed by four-digit value, e.g. O1234. Subroutine No. must start from the top line. The program No. is the subroutine entry address used to call subroutine. The program No. can be in any value, but can’t repeat.

When PMC is executing main program and encounters subroutine calling instruction (CALL), PMC will call and execute the subroutine directly, and return to previous program when the subroutine executes the return instruction (RETURN). To send parameters to subroutine or return, please use the local variables in L register area to prevent the variable values modified by other tasks.

When a main program calls a subroutine, this subroutine can nest and call itself or another subroutine, and allow up to 100 levels of subroutine nesting.

A typical subroutine calling demonstration code follows:

```
    #E100=0                 // initialization parameter
    CALL O123               // call subroutine O123
    EXIT                    // exit main program

    O123                    // subroutine name
```

```
#E100=#E100+1            // accumulated calling times
IF(#E100<10)
    CALL   O123          // nest calling self
RETURN                       // return to previous program
```

When the processing program needs to run same track for several times, please edit this track into subroutine and save in the program storage of the machine tool, and this subroutine can be called if this track should be executed in the program every time.

7.1.6 Motion instruction

Generally, the motion feed of CNC machine tool can be classified into quick positioning feed and working feed.

1) Quick positioning feed instruction

Format: MOVE/G00 X_Y_Z_A_B_C

MOVE and G00: the two instruction keywords have exactly the same function.

X_Y_Z_: target coordinate value of every motion axis; determine whether absolute position value or increment position value according to the mode value of current coordinate system

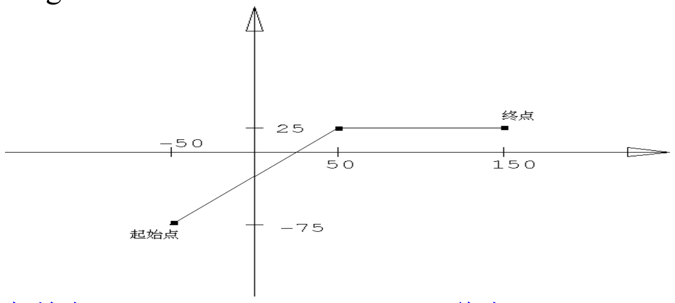
Function: quick positioning instruction makes current motion mode become the quick positioning feed mode of independent motion of every axis, and the feed speed is set by system motor configuration parameter register (Dx15, Dx16). During quick positioning feed, the motion positions of every axis in feeding do not have relationship. They move at the quick shift speed and acceleration/deceleration set by the parameter registers respectively, and are controlled by current F value. Generally, the tool track is a straight line, broken line or irregular arc.

When encounter continuous quick positioning feed instruction, PMC has three operating modes, which are determined by register D38.

- When D38 is 0, if there is new quick shift instruction before previous instruction completes, the system will report error.
- When D38 is 1, if there is new quick shift instruction before previous instruction completes, the system will wait the previous instruction to complete and then execute the new instruction.
- When D38 is 2, if there is new quick shift instruction before previous instruction completes, the system will check whether the axes conflict; if not, execute current quick shift instruction immediately; if yes, wait the previous instruction to complete and then execute the new instruction.

MOVE/G00 programming example:

The position of starting point is X-50, Y-75., instruction G00 X150. Y25.; the tool may run the following track:



起始点 Starting point 终点 Ending point

2) Working feed instruction

Cutting feeding instruction includes linear interpolation and arc interpolation. The axes in feed have interpolation relationship, and their motion combination is cutting feeding motion. The speed control of cutting feeding has three modes, which is determined by the content of register D39. When D39 is 0 or 1, the cutting feeding speed is specified by the F address in the instruction, and the default unit is mm/min. In processing program, F is a mode value, i.e. originally programmed F value is always valid before new F value is specified. When PMC system is just electrified, the F value is specified by register D41. The maximum value of F is controlled by register D42. If the programmed F value is larger than this value, the actual feeding cutting speed will be maintained at this value.

■ When D39 is 0, every motion has independent acceleration/deceleration process, next interpolation instruction will wait previous instruction to complete and then execute; the system software will analyze and execute interpolation instructions one by one.

■ When D39 is 1, the interpolation will execute constant speed motion; the system software will save the analyzed interpolator instructions in hardware interpolation cache and the hardware interpolator executes automatically; the system software won't wait previous instruction to complete and execute later interpolation instructions directly.

■ When D39 is 2, the system will use look-ahead arithmetic, adjust feed speed automatically according to feed track and ending, and the maximum speed is determined by register D41. When executing look-ahead arithmetic, the system will analyze track segment of specified quantity (in register D7) in advance, and the motion starts when reaching specified quantity or encountering WAIT instruction or reaching program end.

Note: at present, look-ahead arithmetic only supports random three axes feed mode; if more than three axes are in motion, the system will report error and stop.

Linear interpolation instruction

Format: LINE/G01 X_Y_Z_A_B_C_F_

LINE and G01: the two instruction keywords have exactly the same function.

X_Y_Z_A_B_C: coordinate values; determine whether absolute position value or increment position value according to the mode value of current coordinate system

F: instruction following speed

Function: LINE/G01 instruction makes current motion mode become linear interpolation mode, the tool moves to specified position to current position, its track is a straight line, F- specifies the speed of the tool in linear motion, and the unit is mm/min.

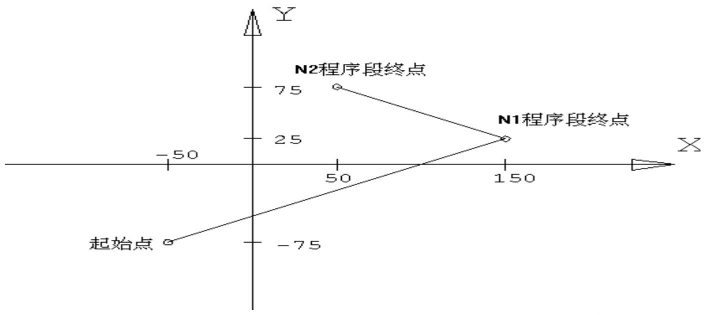
LINE/G01 program example:

Suppose that the position of current tool is X-50. Y-75., and the block follows

N1 G01 X150. Y25. F100;

N2 X50. Y75.;

The tool will run the track shown in the figure below.



N2 程序段终点 N2 block ending point N1 程序段终点 N1 block ending point
起始点 Starting point

Arc interpolation instruction

The instructions below can make the tool move in arc track:

In X—Y plane

G17 {G02/G03} X__ Y__ {(I__ J__)/R__} F__

In X--Z plane

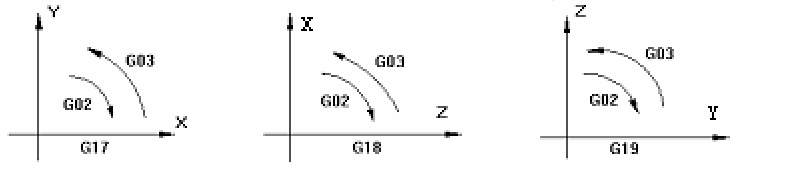
G18 {G02/G03} X__ Z__ {(I__ K__)/R__} F__

In Y--Z plane

G19 {G02/G03} Y__ Z__ {(J__ K__)/R__} F__

S/N	Data content		Instruction	Meaning
1	Plane selection		G17	Specify arc interpolation on plane X--Y
			G18	Specify arc interpolation on plane Z--X
			G19	Specify arc interpolation on plane Y--Z
2	Arc direction		G02	Arc interpolation in CW direction
			G03	Arc interpolation in CCW direction
3	Ending point position	G90 mode	Two axes instruction in X, Y, Z	Coordinates of ending point in current workpiece coordinate system
		G91 mode	Two axes instruction in X, Y, Z	Distance from starting point to ending point (directional)
4	Distance from starting point to circle center		Two axes instruction in I, J, K	Distance from starting point to circle center (directional)
	Arc radius		R	Arc radius
5	Feed rate		F	Motion speed along arc

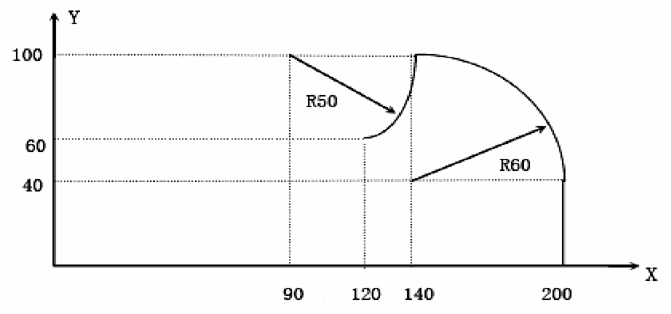
Here, for plane X—Y, the arc direction is the direction of plane X--Y from positive direction of Z axis to negative direction of Z axis; similarly, for plane X—Z or plane Y—Z, the direction is from the positive direction of Y axis or X axis to the negative direction of Y axis or X axis (right hand coordinate system is shown below).



The ending point of the arc is determined by addresses X, Y and Z. In G90 mode, i.e. absolute value mode, addresses X, Y, Z specify the coordinates of arc ending point in current coordinate system; in G91 mode, i.e. increment value mode, addresses X, Y, Z specify the distance between the position of current tool and the ending point in every coordinate axis direction.

In X direction, address I specifies the distance between the position of current tool and the circle center; in Y and Z direction, the distance between the position of current tool and the circle center is specified by J and K, and the symbols of I, J, K values are determined by their motion direction.

To program an arc, in addition to specifying end position and circle center position, we can also program an arc by specifying radius and circle center, specify the radius with address R, and replace the address of specified circle center position. The value of R may be positive and negative. A positive R value is used to program an arc less than 180°, and a negative R value is used to program an arc bigger than 180°. A full circle only can be programmed by specifying the circle center.



Above tracks are programmed in absolute value mode and increment mode respectively:

(1) Absolute value mode

G00 X200.0 Y40.0 Z0

G90 G03 X140.0 Y100.0 I-60.0 F300.0

G02 X120.0 Y60.0 I-50.0

or

G00 X200.0 Y40.0 Z0

G90 G03 X140.0 Y100.0 R60.0 F300.0

G02 X120.0 Y60.0 R50.0

(2) Increment mode

G91 G03 X-60.0 Y60.0 I-60.0 F300.0

G02 X-20.0 Y-40.0 I-50.0

or

G91 G03 X-60.0 Y60.0 R60.0 F300.0

G02 X-20.0 Y-40.0 R50.0

The feed speed of arc interpolation is specified with F, which is the speed of the tool in arc tangent direction.

7.1.7 Other motion related instructions

Pause instruction

Format: PAUSE

No parameter

Function: pause the motion in certain deceleration rate, all motion instructions won't be lost, and will continue running when there is resume instruction. This command is equal to D33=1.

Resume instruction

Format: RESULT

No parameter

Function: resume the paused motion at certain acceleration rate. This command is equal to D33=0.

Emergency stop instruction
Format: STOP
No parameter

Function: stop the executing motion immediately, delete the motion instructions that haven't been executed and exit the task that is executing the motion. Note: in motion state, emergency stop has significant impact on the machine, and may damage the machine easily. To stop the motion task in other states, please pause first and then use the STOP instruction to stop the motion. This command is equal to D32=1, and D32 register resets automatically after emergency stop.

Wait motion complete instruction
Format: WAIT
No parameter

Function: after motion instruction, if it is necessary to wait the motion to complete before executing next instruction, please use WAIT instruction to avoid checking the motion state repeatedly and reducing the efficiency.

DELAY instruction
Format: DELAY(n)

Function: insert n ms delay in current instruction position, and the range of n is 1-65535

Plane selection instruction
G17…… select plane XY
G18…… select plane ZX
G19…… select plane YZ

This group of instructions is used to select the plane of arc interpolation. Usage method:
Linear motion instruction doesn't have relationship with plane selection.
For plane selection related to instruction, please refer to the content about arc interpolation instruction.

Return to reference point instruction
Format: HOME(n,n,n,n,n,n,)

The position of parameter n corresponds to axis 1-6, and the range is 0 or 1,
Function: if the value of every parameter n is 0: the axis won't return to reference point; 1: the axis will return to reference point.
The machine tool coordinate system is established through the operation of returning to reference point after the system is electrified every time. The reference point is a fixed point on the machine tool, and its position is determined by the position sensor of every axis and the home position of every servo motor. When the machine tool returns to reference point, the coordinates of the reference point are 0 in the machine tool coordinate system.
This instruction makes every axis return to reference point of the machine tool from current position at the speed specified by the system. Generally, before using this instruction, please move the workpiece out of the processing area with other motion instructions first and then execute the instruction to return to reference point to prevent the tool knocking into the workpiece.
The speed that every axis returns to the reference point is determined by system register **Dx05, Dx06, Dx07, and Dx11**, which are starting speed, maximum speed, scanning speed and acceleration; **Dx13** and **Dx14** are the maximum travel and maximum scanning travel of

returning to reference point. If current position is near reference point (specified in **Dx35** register), to avoid inconsistent with daily home position due to unexpected shift of current position and entering scope of the reference point, the system will move certain distance to positive direction automatically at half speed. This distance is specified in **Dx34**, and then executes home operation.

The direction of every axis returning to the reference point relates to current machine tool coordinates. If the current position of the axis is positive, return to negative direction; if current position of the axis is negative, return to positive direction. If the machine tool coordinates are set manually before returning to reference point, the direction of returning to reference point can be changed. Several modes are available in the process of returning to reference point, and mainly depend on whether STOP0 and STOP1 signals are valid. Please refer to the introduction below for details.

Set STOP0 signal instruction

Format: SET_STOPA (n,n,n)

The first parameter is specified physical axis

The second parameter is signal enable symbol

The third parameter is signal effective voltage level

Function: STOP0 signal is usually for the sensor signal of external reference point; if STOP0 setting is valid, when the motor runs to reference point position, the axis driving stops immediately when encounters effective trigger voltage level. If STOP0 is invalid, PMC internal software checks the reference point signal automatically when returning to reference point; if reference point signal is valid, decelerate to stop. If STOP1 is invalid, PMC driving axis scans STOP0 in low speed and stops when there is valid signal, returns to reference point, and then PMC sets STOP0 to invalid automatically. If the speed of returning to reference point is high and the inertia is large, it may cause mechanical impact when STOP0 is valid. If this command is used when the program is running, it can be used as hardware limit signal.

Set STOP1 signal instruction

Format: SET_STOPB (n,n,n)

The first parameter is specified physical axis

The second parameter is signal enable symbol

The third parameter is signal effective voltage level

Function: STOP1 signal is usually used as external servo encoder Z phase signal. If STOP1 setting is valid, the motor runs to reference point position and stops when encounters STOP0, and then scans STOP1 signal at low speed. If there is valid trigger voltage level, the axis driving stops immediately and returns to reference point. If STOP1 is invalid, it stops when encounters STOP0, and then scans STOP0 in low speed. If the reference point signal is valid, it stops immediately, and the returning completes.

Set hardware limit signal instruction

Format: SET_LIMIT (n,n,n,n)

The first parameter is specified physical axis

The second parameter is positive limit signal enable symbol, of which 0 indicates valid and 1 indicates invalid

The third parameter is negative limit signal enable symbol, of which 0 indicates valid and 1 indicates invalid

The third parameter is limit signal effective level, of which 0 indicates low level and 1 indicates high level

Function: hardware limit is usually used to prevent the device exceeding the valid motion range unexpectedly during running; if hardware limit signal is valid, the motion stops immediately. If doesn't want to use hardware limit signal, the user can use PLC program to achieve software limit.

7.1.8 IO operation instruction

PMC system provides 80 physical input IO points and 24 physical output IO points, and certain IO points have dedicated function but won't affect the operation of IO instruction. All input/output IO points have internal cache to save the status of last operation.

Read input IO signal instruction

Format: IN (n)

Parameter n is specified input IO label, the range of which is 0-65535

Function: read the status of specified physical input IO or virtual input IO

- If n is 0-79, the status of real physical input IO is read, and the read state is saved in IO cache
- If n is 80-65535, the status of virtual physical input IO (i.e. input IO cache) is read

Actually, the IO address in IN instruction corresponds to the bit address in I register area. I register area is read-only, and can't be rewritten by register evaluation. Virtual input IO extends the read-only bit address in PMC to internal IO in program mode to strengthen the program processing. Virtual input IO has dedicated command to write value. The addressing range of the virtual input IO can be accessed with function code 0x02 in Modbus protocol.

Write virtual input IO signal instruction

Format: WVIN (n, n)

The first parameter n is specified virtual input IO label, the range of which is 0-65535

The second parameter n is the write value, and the range is 0 or 1.

Function: write virtual input IO is to write input IO cache actually. Its actual address corresponds to the bit address in I register area. Except WVIN command, I register area can't be rewritten in any mode.

Write output IO instruction

Format: OUT(n,n)

The first parameter n is specified output IO label, the range of which is 0-65535

The second parameter n is the write value, and the range is 0 or 1.

Function: rewrite the status of specified physical output IO or virtual output IO

- If n is 0-23, the status of real physical output IO is rewritten, and the write state is saved in IO cache
- If n is 24-65535, the status of virtual physical output IO (i.e. output IO cache) is rewritten

Actually, the IO address in OUT instruction corresponds to the bit address in E register area, which is readable/writable, and can be rewritten by register evaluation or OUT() evaluation. In PMC, the bit address in E register area is rewritten in virtual output IO mode. The addressing range of the virtual output IO can be accessed with function code 0x05, 0xf in Modbus protocol.

Read output IO status instruction

Format: ROUT(n)

Parameter n is specified output IO label

Function: read cache of output IO. This command is similar to the function code 0x01 in Modbus protocol, which can read multiple output IO status values continuously.

7.2 Mathematical operation instruction

PMC supports calculation expressions constituted by variables and constants.

Instruction format:

Expressions may be any combination of constants, variables, functions or subexpressions.

Calculation method	Example expression	Description
Assignment	#E100=123	Register assignment
+	#E100=321+123 #E100=#E100+123 #E100=#E101+#E102	Addition
-	#E100=123-321 #E100=#E100-123 #E100=#E101-#E102	Subtraction
*	#E100=#E101*#E102	Multiplication
/	#E100=#E101/#E102	Division
%	#E100=#E101 % #E102	Remainder calculation
&	#E100=#E100 & #E101	Bitwise AND
|	#E100=#E100 | #E101	Bitwise OR
^	#E100=#E100 ^ #E101	Bitwise XOR
~	#E100=~(#E100)	Bitwise NOT
&&	#E100=#E100 && #E101	Logical AND
||	#E100=#E100 || #E101	Logical OR
^^	#E100=#E100 ^^ #E101	Logical XOR
!	#E100=!(#E100)	Logical NOT

<<	#E100=#E100<<8	Left shift operation
>>	#E100=#E100>>8	Right shift operation
Function calculation	#F100=SIN(#E100) #F100=COS(#E100) #F100=TAN(#E100) #F100=ASIN(#E100) #F100=ACOS(#E100) #F100=ATAN(#E100) #F100=SQRT(#E100) #E100=ABS(#S100) #F100=LN(#E100) #F100=EXP(#E100) #F100=LOG(#E100)	Sine calculation Cosine calculation Tangent Arcsine Arc tangent Arc cosine Square root Absolute value Natural logarithm Exponent with e(=2.718…) as the base Logarithm with 10 as the base

<
>
<=
>=
==
!=
	IF(#E100<100)
	IF(#E100)100)
	IF(#E100<=100)
	IF(#E100>=100)
	IF(#E100==100)
	IF(#E100!=100)
		Operator of less than
		Operator of larger than
		Operator of less than or equal to
		Operator of larger than or equal to
		Operator of equal to
		Operator of not equal to

Combined expression	#F100=(((#E100+2)/3)-5)*SIN(30)	Multiple nested mathematical expressions

Expression calculation priority:

Priority Lower level indicates higher priority	Calculation symbol
1	#
2	( )
3	Function (SIN,COS,EXP…)
4	*,/,%
5	+,-
6	<<,>>
7	&,|,~,^
8	<,>,<=,>=,==,!=
9	&&,||,^^,!
10	=

- The calculation expressions of same priority shall follow the principle of from left to right.
- Calculation expressions have multiple levels of priority. The calculation expression is long, please force the priority with ().
- () can be nested in the calculation for up to five levels.

Calculation accuracy

The value of macro variable has seven significant digits, and therefore the accuracy will be reduced if the value is too large or too small (9999999.000~0.0000001), and repeated calculation will accumulate the error. Therefore, the variable values should be in a reasonable range;

7.1.10 Other system instructions

Set IP address instruction

Format: SET_IP n“xxx.xxx.xxx.xxx”

The first parameter n is the address type, the range of which is 0-2; 0: IP address of PMC, 1: intranet mask, 2: gateway address.

The second parameter is dotted string type.

- ■ If n=0, the parameter of the second string will be IP address, e.g. “192.168.0.123”
- ■ If n=1, the parameter of the second string will be intranet mask, e.g. “255.255.255.0”
- ■ If n=2, the parameter of the second string will be gateway address, e.g. “192.168.0.1”

Function: set IP address, subnet mask and gateway address of PMC.

The default IP address of PMC is “192.168.0.123”; to connect PMC to the LAN, the IP address must be unique in the LAN, and the IP address and mask should match the LAN.

SET_IP instruction actually writes above parameters into C40, C41, and C42 registers, and the saved system parameters take effect after restarted.

Set NIC address instruction

Format: SET_MAC “xx-xx-xx-xx-xx-xx”

The address parameter is a string; x is hexadecimal character, and the range is 0-9, A-F; two xx combine a section, and there are six sections totally. Every section is separated with ‘-’. For example: “11-22-33-44-55-AA”

Function: set the NIC address MAC of PMC; to connect PMC to LAN, the MAC address must be unique in the LAN.

SET_MAC instruction actually writes above parameters into C43-C44 registers, and the saved system parameters take effect after restarted.

Set serial port instruction

Format: SET_UART(n,n,n,n,n)

The first parameter n is serial port No., the range of which is 0-1,

The second parameter n is serial baud rate, 9600/ 14400/ 19200/ 38400/ 56000/ 57600/ 115200 optional, default: 115200

The third parameter n is valid data bit width of serial transmission, and the virtual value is fixed at 8

The fourth parameter n is the parity mode of serial transmission, and the virtual values are 0, 1, 2, indicating No, odd, and even respectively.

The fifth parameter n is the stop bit width of serial transmission, and the virtual value is 0 or 2, 0: stop bit width is 1, 2: stop bit width is 2, 1: stop bit width is 1.5 (unavailable at present); default: 0.

Function: set serial communication parameters.

serial port 0 is used for PMC console operation, and serial port 1 is used for Modbus data communication. Actually, these parameters will be written into C0-C3 (serial port 0) or C11-C14 (serial port 1) registers. Serial parameter setting takes effect immediately, but hasn't been saved in system parameter file. If the parameter setting isn't proper, the system will restore original setting after restarted. If the user confirms that the parameter configuration is appropriate, please save with the SAVE_SYS command.

Get system communication interface info
Format: IPCONFIG
No parameter
Function: show network configuration info and serial port configuration info in the console

Set system date instruction
Format: SET_DATE(n,n,n,n)
The first parameter n is the year
The second parameter n is the month
The third parameter n is the week day
The fourth parameter n is the date
Instruction example: SET_DATE(2011,4,5,1) indicates that current date is Friday, April 1st, 2011

Set system time instruction
Format: SET_TIME(n,n,n)
The first parameter n is the hour
The second parameter n is the minute
The third parameter n is the second
Instruction example: SET_TIME(11,30,25) indicates that current time is 11:30:25

Get system date instruction
Format: DATE
No parameter
Function: get current date from the console

Get system time
Format: TIME
No parameter
Function: get current time from the console

Save system parameters

Format: SAVE_SYS n

Parameter n is saving mode, the range of which is 1, 2, 3, 5, 6, 7

Function: the system registers in key areas will be saved in ferroelectric RAM (Nonvolatile Memory) or ROM. The saved register areas are I0-I127, E0-E127, S0-S127, C0-C127, D0-D700, F0-F127.

- If n=1, the system parameters will be saved in ferroelectric RAM.
- If n=2, the system parameters will be saved in ROM.
- If n=3, the system parameters will be saved in ferroelectric RAM and ROM.
- If n=5, the system parameters and current coordinates of the machine tool will be saved in ferroelectric RAM.
- If n=6, the system parameters and current coordinates of the machine tool will be saved in ROM.
- If n=7, the system parameters and current coordinates of the machine tool will be saved in ferroelectric RAM and ROM.

Load system parameters

Format: LOAD_SYS n

Parameter n is loading mode, the range of which is 1, 2, 5, 6

Function: load the saved data from ferroelectric RAM or ROM into key register area, which may be I0-I127, E0-E127, S0-S127, C0-C127, D0-D700, F0-F127.

- If n=1, the system parameters will be loaded from ferroelectric RAM.
- If n=2, the system parameters will be loaded from ROM.
- If n=5, the system parameters and current coordinates of the machine tool will be loaded from ferroelectric RAM.
- If n=6, the system parameters and current coordinates of the machine tool will be loaded from ROM.

Get system software version

Format: VERSION

No parameter

Function: show current PMC software version in console.

8 Online Development

PMC system opens its internal register area to any third party software or controller, accesses these registers through standard MODBUS communication protocol and executes related control functions. PMC supports R232 serial based and Ethernet media based MODBUS communication, as well as ASIC and RTU data stream modes; Ethernet media based MODBUS communication also supports TCP/IP or UDP protocol mode.

This PMC system development kit also provides upper computer API function library based on Windows-PC online development mode, and corresponding engineering DEMO and related tools. Actually, most of these API functions are achieved by operating the system registers in PMC through Modbus protocol, and we also provides basic functions to operate these registers, so that the users can extend the functions according to actual requirements and improve system flexibility.

The development process connected to third party controller depends on the software development environment of the third party controller. Generally, if only it supports standard R232 or Ethernet Modbus protocol, it is possible to access with the0x1, 0x2, 0x03, 0x04, 0x5, 0x06, 0xf, 0x10 function code in Modbus protocol according to the system register address and offset described in Chapter 5, and thus control the PMC. This Manual only describes the development method based on Windows-PC online mode.

8.1 Basic library functions list

Category	Function name	Function	Page
Basic parameters	pmc_modpara_init	Parameter initialization function	40
	pmc_serial_init	Serial port initialization	40
	pmc_set_signle_para	Set basic parameters	40
	pmc_net_conn	TCP/UDP connection	41
	pmc_clos_netdev	Close communication port	41
	pmc_set_consleport	Set console command communication port	41
	pmc_uart_recv	Serial port read data function	41
	pmc_get_consleport	Get console command communication port	42
	pmc_close_port	Close corresponding connection port	42
	pmc_set_stop0_mode	Stop mode	42
	pmc_set_stop1_mode	Stop mode	42
	pmc_set_limit_mode	Limit mode	43
	pmc_set_pulse_mode	Pulse mode	43
Driving status check	pmc_get_run_status	Get system driving status	44
	pmc_get_back_status	Get home function status	45
	pmc_get_axis_status	Get single axis status	45
	pmc_get_inp_status	Get interpolation drive status	45
Motion parameter setting	pmc_set_speed	Set single axis motion parameters	46
	pmc_set_speedExt	Set multiple axes motion parameters	46
	pmc_set_command_pos	Set the value of logic position counter of every axis	46

8.2 Details of basic library functions

8.2.1 Communication interface category

pmc_modpara_init ( )

Function name	int pmc_modpara_init(void)
Description	Device interface parameter initialization function
Input parameter	None
Output parameter	None
Return	0: normal, -1: send failed
Note	Initialization should be called first before using motion function

pmc_serial_init ( )

Function name	int pmc_serial_init(INT8U UartPort, INT32U UartBaud, INT8U DataBit, INT8U StopBit, INT8U Parity, BOOL Flag=TRUE)
Description	Initialize serial port
Input parameter	UartPort: serial port (0,1,2,3) UartBaud: set the baud rate corresponding to serial port DataBit: serial port data bit (5, 6, 7, 8 bits) StopBit: serial port stop bit (1, 2-bit) Parity: serial port parity bit (no, odd, even) Flag: whether this serial port is dedicated for Modbus communication TRUE: Modbus communication (default), FALSE: for common serial port (i.e. for console communication)
Output parameter	None
Return	0: normal, -1: error
Note	Network connection should be called before using motion function

pmc_set_signle_para()

Function name	int pmc_set_signle_para(INT8U port, INT32U mode)
Description	Set basic parameters
Input parameter	port: corresponding communication port No. mode: communication mode of writing data and data type configuration parameter; default data access mode: readable/writable
Output parameter	None

Return	0: normal, -1: error

pmc_net_conn ( )

Function name	int pmc_net_conn(INT8U NetPort, INT32S iDevNum, INT32U uiConIp, INT16U nConPort)
Description	Connect TCP/UDP of Modbus, and map with device No. and IP address respectively
Input parameter	NetPort: network port No., only the port No. for TCP and UDP connection, uiConIp: connected server IP address nConPort: connected server communication port No. (Modbus communication is fixed at 502 generally)
Output parameter	None
Return	0: normal, -1: error

pmc_clos_netdev ( )

Function name	int pmc_clos_netdev(INT8U NetPort, INT32S iDevNum)
Description	Close the communication port
Input parameter	NetPort: communication port No. (only valid for TCP/UPD network port) iDevNum: device No. corresponding to network communication
Output parameter	None
Return	0: normal, -1: error

pmc_set_consleport ( )

Function name	int pmc_set_consleport(INT8U Port)
Description	Set console command communication port
Input parameter	Port: the set port No.
Output parameter	None
Return	1: true, 0: false

pmc_uart_recv ( )

Function name	int pmc_uart_recv(INT8U Port, INT8U *rbuff, INT16U length)
Description	Serial port reading data function
Input parameter	Port: communication port No. *rbuff: storage area of read data length: specify the length of read data
Output parameter	None
Return	the length of received data

pmc_get_consleport ( )

Function name	int pmc_get_consleport(void)
Description	get console command communication port No.
Input parameter	None
Output parameter	None
Return	the set console command communication port No.

pmc_close_port ( )

Function name	int pmc_close_port(INT8U Port)

Description	Close the corresponding connection port of Modbus
Input parameter	Port: communication port No. (including serial port and network port)
Output parameter	None
Return	0: normal, -1: error

8.2.2 Driver status check category

pmc_get_run_status ( )

Function name	int pmc_get_run_status(INT32S id, INT32U *status)
Description	Get system drive status
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission
Output parameter	status: motion status read of the system (0: idle, 1: in motion)
Return	0: execute successfully, -1: send failed
Note	None

pmc_get_back_status ( )

Function name	int pmc_get_back_status(INT32S id, INT32U *status);
Description	Get the status of home function
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission
Output parameter	status: home status read of the system
Return	0: execute successfully, -1: send failed
Note	None

pmc_get_axis_status()

Function name	int pmc_get_axis_status(INT32S id, INT16S *status1, INT16S *status2, INT16S *status3, INT16S *status4, INT16S *status5, INT16S *status6);
Description	Get axis status
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission
Output parameter	status: read axis motion status
Return	0: execute successfully, -1: send failed
Note	None

pmc_get_inp_status( )

Function name	int _stdcall pmc_get_inp_status(INT32S id, INT32U *status)
Description	Get drive status of interpolation

Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission
Output parameter	status: interpolation status read of the system (0: idle, 1: interpolating)
Return	0: execute successfully, -1: send failed
Note	None

8.2.3 Motion parameter setting category

pmc_set_speed ( )

Function name	int pmc_set_speed(INT32S id, INT16U axis, INT32S sv, INT32U speed, INT32U acc, INT32U pmm)
Description	Set the parameters of every axis
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission axis: axis number (1 - 6) sv: axis starting speed speed: axis driving speed acc: axis acceleration pmm: axis pulse equivalent numerator
Output parameter	None
Return	0: execute successfully, -1: send failed

pmc_set_speedExt( )

Function name	int pmc_set_speedExt(INT32S id, INT16U *axis, INT32U *sv, INT32U *speed, INT32U *acc, INT32U *pmm, INT16U num);
Description	Set parameters of multi-axis
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission *axis: save the array of axis number (1-6) *sv: save the array of axis starting speed *speed: save the array of axis driving speed *acc: save the array of axis acceleration *pmm: save the array of axis pulse equivalent numerator
Output parameter	None
Return	0: execute successfully, -1: send failed

pmc_set_command_pos ( )

Function name	int pmc_set_command_pos(INT32S id, INT16U axis, INT32S pulse);
Description	Set the logical position counter value of every axis

Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission axis: axis number (1 - 6) pulse: set pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455)
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	The logical position counter can be written and read at any time

pmc_set_command_posExt ( )

Function name	int pmc_set_command_posExt(INT32S id, INT16U *axis, INT32S *pulse, INT16U num);
Description	Set the logical position counter value of multi-axis
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission *axis: save the array of axis number (1-6) *pulse: save the array of set pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455) Num: corresponding axis number
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	The logical position counter can be written and read at any time

pmc_set_actual_pos ( )

Function name	int pmc_set_actual_pos(INT32S id, INT16U axis, INT32S pulse);
Description	Set the value of actual position counter
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission axis: axis number (1 - 6) pulse: set pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455)
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	The actual position counter can be written and read at any time

pmc_set_actual_posExt( )

Function name	int pmc_set_actual_posExt(INT32S id, INT16U *axis, INT32S *pulse, INT16U num);
Description	Set the value of actual position counter

Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission *axis: axis number (1 - 6) *pulse: set pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455) Num: corresponding axis number
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	The actual position counter can be written and read at any time

pmc_coord_mode( )

Function name	int pmc_coord_mode(INT32S id, INT32U mode)
Description	Set current coordinate mode of current task
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission mode: 0: increment coordinate mode, 1: absolute coordinate mode
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	None

pmc_multaxis_conftmode( )

Function name	int pmc_multaxis_conftmode(INT32S id, INT32U conftmode)
Description	Multi-axis motion conflict treatment
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission conftmode: 0: pause and report error 1: execute new motion when the system is in idle 2: if there is no axis conflict in non-interpolation state, allow executing multi-axis linkage again
Output parameter	None
Return	0: execute successfully, -1: send failed

pmc_command_executmode ( )

Function name	int pmc_command_executmode(INT32S id, INT32U executmode)
Description	Motion instruction execution mode

Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission executmode: 0 – execute current motion instruction in real-time and then execute next motion instruction; every motion instruction has independent acceleration / deceleration segment; 1 – send continuous motion instruction to hardware cache, and then hardware executes continuous interpolation motion; every motion instruction is at constant speed, and there is no acceleration / deceleration segment; 2 – Smooth the continuous motion instruction according to its motion track. Default: 0
Output parameter	None
Return	0: execute successfully, -1: send failed

pmc_set_stop0_mode ( )

Function name	int pmc_set_stop0_mode(INT32S id, INT16U axis,INT16U value,INT16U logic)
Description	Set valid/invalid and logical voltage level of stop0 signal
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission axis: axis number (1 - 6) value: 0: invalid 1: valid logic: 0: low voltage level effective 1: high voltage level effective
Output parameter	None
Return	0: send successfully, -1: send failed

pmc_set_stop1_mode ( )

Function name	int pmc_set_stop1_mode(INT32S id, INT16U axis,INT16U value,INT16U logic)
Description	Set valid/invalid and logical voltage level of stop1 signal
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission axis: axis number (1 - 6) value: 0: invalid 1: valid logic: 0: low voltage level effective 1: high voltage level effective
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	When initializing, the state is signal invalid, stop at low voltage level The stop mode is immediate; STOP1 signal is also the same

pmc_set_limit_mode ( )

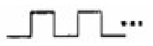
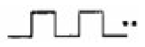
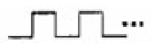
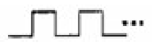
Function name	int pmc_set_limit_mode(INT32S id, INT16U axis,INT16U v1,INT16U v2,INT16U logic)
Description	Set the mode of positive/negative limit input nLMT signal

Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission axis: axis number (1 - 6) v1: 0: invalid 1: valid v2: 0: negative limit valid 1: negative limit invalid logic: 0: low voltage level effective 1: high voltage level effective
Output parameter	None
Return	0: execute successfully, -1: send failed, -4: invalid device interface
Note	The initial state is positive/ negative limit low voltage level valid

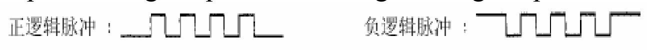
pmc_set_pulse_mode ( )

Function name	int pmc_set_pulse_mode(INT32S id, INT16U axis,INT8U value,INT8U logic,INT8U dir_logic)
Description	Set the work mode of output pulse
Input parameter	dev_num: device No. axis: axis number (1 – 6) value: 0: pulse + pulse 1: pulse + direction

Both pulse and direction are positive logic setting

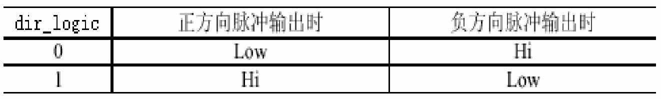
Pulse output mode	Driving direction	Output signal wave	
		PU/CW signal	DR/CCW signal
Independent two pulses mode	+ direction driving output		Low level
	- direction driving output	Low level	
One pulse mode	+ direction driving output		Low level
	- direction driving output		High level

logic: 0: positive logical pulse 1: negative logical pulse



正逻辑脉冲 Positive logic pulse \ 负逻辑脉冲 Negative logic pulse

dir-logic: 0: direction output signal is positive logic
1: direction output signal is negative logic



dir_logic	正方向脉冲输出时	负方向脉冲输出时
0	Low	Hi
1	Hi	Low

正方向脉冲输出时 Positive pulse output \ 负方向脉冲输出时 Negative pulse output

Output parameter	None
Return	0: execute successfully, -1: send failed
Note	The initial state is pulse + direction, positive logic pulse, and direction output signal is positive logic

8.2.4. Motion parameter check category

pmc_get_axis_pos ( )

Function name	int pmc_get_axis_pos(INT32S id, INT16U axis, INT32S *compos, INT32S *actpos, INT32U *speed)

Description	Get the logical and actual position status of every axis
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission axis: axis number (1 - 6)
Output parameter	*compos: read logical position of the axis, range (-268435455~+268435455) *actpos: read actual position of the axis, range (-268435455~+268435455) *speed: read drive speed of the axis, range (-268435455~+268435455)
Return	0: execute successfully, -1: send failed
Note	This function can get the logical position, actual position and driving speed of the axis at any time, and can indicate the current position of the axis when the motor isn't out of step

pmc_get_command_pos ( )

Function name	int pmc_get_command_pos(INT32S id, INT32S *compos, INT32S *actpos, INT32U *speed)
Description	Get all logical position
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission
Output parameter	*compos: read logical position of the axis, range (-268435455~+268435455) *actpos: read actual position of the axis, range (-268435455~+268435455) *speed: read drive speed of the axis, range (-268435455~+268435455)
Return	0: execute successfully, -1: send failed
Note	This function can get the logical position, actual position and driving speed of the axis at any time, and can indicate the current position of the axis when the motor isn't out of step

pmc_get_register_pos( )

Function name	int pmc_get_register_pos(INT32S id, INT32U *regpos)
Description	Get or set the axis logical position of the coordinate system specified in D4 register
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission
Output parameter	regpos: the axis logical position of the coordinate system specified in the register
Return	0: execute successfully, -1: send failed
Note	None

8.2.5. Driver category

pmc_pmove ( )

Function name	int pmc_pmove(INT32S id, INT16U axis, INT32S pulse)
Description	Quantitative drive

Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission axis: axis number (1 - 6) pulse: output pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455)
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	The parameters of speed curve must be set properly before writing drive command

pmc_pmoveExt( )

Function name	int pmc_pmoveExt(INT32S id, INT16U *axis, INT32S *pulse, INT16U num)
Description	Multi-axis quantitative drive
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission *axis: save the array of axis number (1-6) *pulse: output pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455) Num: axis number
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	The parameters of speed curve must be set properly before writing drive command

pmc_continue_move( )

Function name	int pmc_continue_move(INT32S id, INT16U axis, INT16S dir)
Description	Continuous drive of every axis
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission axis: axis number (1 - 6) dir: running direction, 0: positive, non-zero: negative
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	The parameters of speed curve must be set properly before writing drive command

pmc_continue_moveExt( )

Function name	int pmc_continue_moveExt(INT32S id, INT16U *axis, INT16S *dir, INT16U num);
Description	Multi-axis quantitative drive

Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission *axis: save the array of axis number (1-6) *dir: save the array of running direction, 0: positive, non-zero: negative Num: axis number
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	The parameters of speed curve must be set properly before writing drive command

pmc_line2( )

Function name	int pmc_line2(INT32S id, INT16U axis1, INT32S pulse1, INT16U axis2, INT32S pulse2)
Description	Two-axis linear interpolation
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission axis1: axis number (1 - 6) pulse1: output pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455) axis2: axis number (1 - 6) pulse2: output pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455)
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	Support random two-axis interpolation, and the interpolation speed bases on the minimum axis speed

pmc_line3 ( )

Function name	int pmc_line3(INT32S id, INT16U axis1, INT32S pulse1, INT16U axis2, INT32S pulse2, INT16U axis3, INT32S pulse3)
Description	Three-axis linear interpolation
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission axis1: axis number (1 - 6) pulse1: output pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455) axis2: axis number (1 - 6) pulse2: output pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455) axis3: axis number (1 - 6) pulse3: output pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455)
Output parameter	None
Return	0: execute successfully, -1: send failed

Note	Support random three-axis interpolation, and the interpolation speed bases on the minimum axis speed

pmc_line4 ( )

Function name	int pmc_line4(INT32S id, INT32S pulse1, INT32S pulse2, INT32S pulse3, INT32S pulse4)
Description	Four-axis linear interpolation
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission pulse1, pulse2, pulse3, pulse4: output pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455)
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	The speed of four-axis interpolation bases on X axis speed

pmc_line5( )

Function name	int pmc_line5(INT32S id, INT32S pulse1, INT32S pulse2, INT32S pulse3, INT32S pulse4, INT32S pulse5)
Description	Five-axis linear interpolation
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission pulse1, pulse2, pulse3, pulse4, pulse5: output pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455)
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	The speed of five-axis interpolation bases on X axis speed

pmc_line6( )

Function name	int pmc_line6(INT32S id, INT32S pulse1, INT32S pulse2, INT32S pulse3, INT32S pulse4, INT32S pulse5, INT32S pulse6)
Description	Six-axis linear interpolation
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission pulse1, pulse2, pulse3, pulse4, pulse5, pulse6: output pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455)
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	The speed of six-axis interpolation bases on X axis speed

pmc_fifo1 ( )

Function name	int pmc_fifo1(INT32S id, INT16U axis1, INT32S pulse1, INT32U speed)
Description	Cache random single axis interpolation motion
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission axis1: axis number (1 - 6) pulse1: output pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455) speed: operating speed value (1 - 2000000)
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	

pmc_fifo2 ( )

Function name	int pmc_fifo2(INT32S id, INT16U axis1, INT32S pulse1, INT16U axis2, INT32S pulse2, INT32U speed)
Description	Cache random two-axis interpolation motion
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission axis1: axis number (1 - 6) pulse1 output pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455) axis2: axis number (1 - 6) pulse2: output pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455) speed: operating speed value (1 - 2000000)
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	

pmc_fifo3()

Function name	int pmc_fifo3(INT32S id, INT16U axis1, INT32S pulse1, INT16U axis2, INT32S pulse2, INT16U axis3, INT32S pulse3, INT32U speed)
Description	Cache random three-axis interpolation motion

Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission axis1: axis number (1 - 6) pulse1: output pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455) axis2: axis number (1 - 6) pulse2: output pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455) axis3: axis number (1 - 6) pulse3: output pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455) speed: operating speed value (1 - 2000000)
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	None

pmc_fifo4 ()

Function name	int pmc_fifo4(INT32S id, INT32S pulse1, INT32S pulse2, INT32S pulse3, INT32S pulse4, INT32U speed)
Description	Cache four-axis interpolation motion
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission pulse1, pulse2, pulse3, pulse4: output pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455) speed: operating speed value (1 - 2000000)
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	None

pmc_fifo5 ()

Function name	int pmc_fifo5(INT32S id, INT32S pulse1, INT32S pulse2, INT32S pulse3, INT32S pulse4, INT32S pulse5, INT32U speed)
Description	Cache five-axis interpolation motion
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission pulse1, pulse2, pulse3, pulse4, pulse5: output pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455) speed: operating speed value (1 - 2000000)

Output parameter	None
Return	0: execute successfully, -1: send failed
Note	None

pmc_fifo6 ()

Function name	int pmc_fifo5(INT32S id, INT32S pulse1, INT32S pulse2, INT32S pulse3, INT32S pulse4, INT32S pulse5, INT32S pulse6,INT32U speed)
Description	Cache six-axis interpolation motion
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission pulse1, pulse2, pulse3, pulse4, pulse5, pulse6: output pulses, >0: motion in positive direction, <0: motion in negative direction Range (-268435455~+268435455) speed: operating speed value (1 - 2000000)
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	None

pmc_go_home()

Function name	int pmc_go_home(INT32S id, INT16U axis);
Description	Home function of every axis
Input parameter	Axis: axis number (1-6)
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	None

8.2.6. Stop category

pmc_stop ()

Function name	int pmc_stop(INT32S id)
Description	Stop processing, and give up current processing task
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	None

pmc_pause ( )

Function name	int pmc_pause(INT32S id)
Description	Pause processing
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	None

pmc_resume( )

Function name	int pmc_resume(INT32S id)
Description	Exit pause and resume processing
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	None

pmc_axis_stop( )

Function name	int pmc_axis_stop(INT32S id, INT16U axis1=0, INT16U axis2, INT16U axis3, INT16U axis4, INT16U axis5, INT16U axis6)
Description	Select an axis to stop
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission axis: axis number (1 - 6)
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	None

8.2.7. Switch quantity I/O category

pmc_read_Inbit( )

Function name	int pmc_read_Inbit(INT32S id, INT16U bit_start, INT16U bit_num, INT8U *status)
Description	Read the status of single input point

Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission bit_start: input port No. bit_num: continuously read input port number
Output parameter	*status: number of saving input status
Return	0: execute successfully, -1: send failed
Note	Refer to the hardware manual of respective controller for the range of input point and corresponding function

pmc_write_Outbit( )

Function name	int pmc_write_Outbit(INT32S id, INT16U bit_start, INT16U bit_num, INT8U *status)
Description	Set single output point
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission bit_start: output port No. bit_num: continuously read input port number
Output parameter	*status: number of saving output status
Return	0: execute successfully, -1: send failed
Note	Refer to the hardware manual of respective controller for the range of output point and corresponding function

pmc_read_Outbit ()

Function name	int adt8860a_sudden_write_bit(int dev_num, int number, int value)
Description	Read the status of single output point
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission bit_start: input port No. bit_num: continuously read input port number
Output parameter	*status: number of saving input status
Return	0: execute successfully, -1: send failed
Note	Refer to the hardware manual of respective control card for the range of output point and corresponding function

8.2.8. System category

pmc_set_cpara( )

Function name	int pmc_set_cpara(INT32S id, INT16U addr, INT32U *data, INT16U len)
Description	Set parameters of system register

Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission addr: register address corresponding to system parameter (32-bit double-word address) *data: save the parameter values to be set len: set parameter length
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	

pmc_get_cpara( )

Function name	int pmc_get_cpara(INT32S id, INT16U addr, INT32U *data, INT16U len)
Description	Get parameters of system register
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission addr: register address corresponding to system parameter (32-bit double-word address) len: get parameter length
Output parameter	*data: save the parameter values
Return	0: execute successfully, -1: send failed
Note	

pmc_set_dpara( )

Function name	int pmc_set_dpara(INT32S id, INT16U addr, INT32U *data, INT16U len)
Description	Set parameters of drive register
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission addr: register address corresponding to system parameter (32-bit double-word address) *data: save the parameter values to be set len: set parameter length
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	

pmc_get_dpara( )

Function name	int pmc_get_dpara(INT32S id, INT16U addr, INT32U *data, INT16U len)
Description	Get parameters of drive register
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission addr: register address corresponding to system parameter (32-bit double-word address) len: get parameter length

Output parameter	*data: save the parameter values
Return	0: execute successfully, -1: send failed
Note	None

pmc_run( )

Function name	int pmc_run(INT32S id);
Description	Write data
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	None

pmc_set_timeout ( )

Function name	int pmc_set_timeout(INT16U timeouts, INT8U repeat_times);
Description	Set the timeout and repeat times of corresponding port
Input parameter	port: communication port No. timeouts: timeout(unit: ms) > 500ms repeat_times: repeat times
Output parameter	None
Return	0: execute successfully, -1: send failed
Note	None

pmc_read_data ( )

Function name	int pmc_read_data(INT32S id, INT16U start_addr, INT8U *pReadBuff, INT16U len);
Description	as client port, read server data function
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission start_addr: starting address of read data len: byte length of read data
Output parameter	pReadBuff: storage area of read data
Return	0: execute successfully, -1: send failed
Note	

pmc_write_data( )

Function name	int pmc_write_data(INT32S id, INT16U start_addr, INT8U *pWriteBuff, INT16U len)
Description	as client port, write data to server function
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission start_addr: starting address of write data pWriteBuff: storage area of write data len: byte length of write data
Output parameter	*pData: ModParam structure pointer
Return	0: execute successfully, -1: send failed
Note	

pmc_rfile_data( )

Function name	int pmc_rfile_data(INT32S id, INT32U FrameCnt, INT8U *pReadBuff, INT16U *pDataSize)
Description	read file data according to frame No.
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission FrameCnt: frame No. of read file data; the maximum byte length of read data once is FileFrameSize
Output parameter	*pReadBuff: storage area of file frame data *pDataSize: size of frame data read once
Return	0: execute successfully, -1: send failed
Note	

pmc_read_file()

Function name	int pmc_read_file(INT32S id, char *pFilePath, INT32U *pFrameTotal, INT32U *pFileSize)
Description	read information of specified file
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission pFilePath: path of read file (absolute path including absolute path)
Output parameter	*pFrameTotal: read frames required for file transmission *pFileSize: read file size
Return	0: normal, -1: send failed
Note	

pmc_write_file()

Function name	int pmc_write_file(INT32S id, char *pFilePath, INT32U pFrameTotal, INT32U pFileSize)
Description	write information of specified file
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission pFilePath: path of written file FrameTotal: frames of file to be transmitted FileSize: size of written file
Output parameter	
Return	0: normal, -1: send failed
Note	

pmc_wfile_data ()

Function name	int pmc_wfile_data(INT32S id, INT32U FrameCnt, INT8U *pWriteBuff, INT16U DataSize)
Description	write file data according to frame No.
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission FrameCnt: frame No. of writing file data; the maximum byte length of write data once is FileFrameSize pWriteBuff: storage area of write file frame data DataSize: size of frame data written once
Output parameter	
Return	0: normal, -1: send failed
Note	

pmc_read_dir()

Function name	int pmc_read_dir(INT32S id, char *pDirPath, CDir *pDirList, INT32U *pDirNum)
Description	read directory info
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission *pDirPath: read directory path *pDirList: save read directory info *pDirNum: save the directory or file number in read directory info
Output parameter	
Return	0: normal, -1: send failed
Note	

pmc_create_dir ()

Function name	int pmc_create_dir(INT32S id, char *pDirPath)
Description	create directory
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission *pDirPath: the directory path to be created
Output parameter	
Return	0: normal, -1: send failed
Note	

pmc_del_dir()

Function name	int pmc_del_dir(INT32S id, char *pDirPath)
Description	Delete directory
Input parameter	id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission *pDirPath: the directory path to be deleted
Output parameter	
Return	0: normal, -1: send failed
Note	

8.3 Motion control function library guide

8.3.1. ADT-8860 function library description

ADT-8860 function library is the interface for users operating motion control card. The users can control the motion card to complete corresponding function by calling interface functions.

8.3.2. Calling DLL in Windows

The DLL “ADT8860.dll” in Windows is written with VC. It is in “Development Kit \Drivers\DLL” in the CD, and is suitable for the programming language tools in Window: VB, VC, etc.

8.3.3. Calling in VC

(1) Create a new item;
(2) Copy the files “ADT8860.lib” and “adt8860.h” from “Development Kit \VC” in the CD to the path of the new item;

(3) In the “File View” of the “Work Area” of the new item, right click the mouse and select “Add Files to Project”, select “Library Files(.lib)” in the Add Files dialog box, search and select “ADT8860.lib” and click “OK” to load the static library;

(4) Add #include “adt8860.h” to the statement of source file or header file or global header filer “StdAfx.h”;

After above four steps, the user can call the functions in the DLL.

Note: The calling in VC.NET is similar as VC.

8.3.4. Calling in VB

(1) Create a new item;

(2) Copy the file “adt8860.bas” from “Development Kit \VB” in the CD to the path of the new item;

(3) Select “Project\Add module” menu, and select the “Save” tab in the dialog box, search the “adt8860.bas” module file, and click the Open button;

After above three steps, the user can call the functions in the DLL.

Note: The calling in VC.NET is similar as VB.

8.3.5 Calling help when use application development

Use serial testing tool to show the actually received input parameters in the control card and the execution result. Serial testing display is enabled or disabled through pmc_uart_recv() parameter, and the default state is off. When serial testing is enabled, it will take up more system resource of the control card, and affect the execution speed, but it will help developers check the reason of control card execution failure caused by illegal parameters.

8.4 Motion control development points

There will be some problems in the programming of the card. In fact, most of the problems are caused by misunderstanding the principal of the control card. Below is the description of some familiar instances that are easily understood.

8.4.1. Card initialization

Call pmc_modpara_init () function at the beginning of the program to initialize the device address, and then check the network connection through pmc_net_conn () function and check whether ADT8860 card is connected properly. If yes, the system will return a device No. Then, set the pulse output mode and limit switch mode according to the configuration of the device. pmc_modpara_init t() should be called only once when the application is initialized. When the application is ended, please call pmc_close_port () function to release all network resource.

Note: library functions “pmc_modpara_init()” and “pmc_net_conn()” are the “door” to ADT8860 card. Calling other functions has meaning only after calling this function and initializing the motion control card successfully.

8.4.2. Axis speed setting

1) Constant speed motion

The parameter setting is simple. It is only required to set the driving speed to be lower than or equal to the start speed, and other parameters do not need setting.

Related function: pmc_set_speed

2) Symmetric linear acceleration/deceleration

This is a most commonly used mode. It requires setting start speed, driving speed and acceleration, and uses automatic deceleration.

Related function: pmc_set_speed

3) Interpolation speed

ADT8860 card can select random 1-3 axes for linear interpolation, and fixed 4-6 axes linear interpolation. For the interpolation speed, the speed parameter of the frontmost axis is used as the speed of long axis, e.g.:

pmc_line2 (0,1,100,2,200)

Uses the speed parameter of the first axis, i.e. X axis, and doesn't relate to the sequence in the parameters.

pmc_line3 (0,1,100,2,200,3,500)

Uses the speed parameter of the second axis, i.e. Y axis, and doesn't relate to the sequence in the parameters.

Note: The speed ratio of interpolation is half of the ratio of single axis motion, that is to say, the interpolating speed is only half of single axis motion under same parameters.

8.4.3 STOP0, STOP1 signal

STOP0 and STOP1 are the signals that every axis has, and thus there are eight STOP signals totally. These signals are mainly used for home operation. Home mode can use one or several signals accordingly. However, please note that this signal is decelerating stop. For high speed home operation, please add a deceleration switch before the home switch, that is to say, use two STOP signals, one home switch and one deceleration switch. It is also possible to use one signal, which decelerates stop after meeting STOP signal, moves reversely at constant speed and stops after meeting again.

8.5 Motion control development programming example

All motion control functions are immediate return. After sending driving command, the motion is controlled by the motion card. At this moment, the user's upper computer software can monitor the entire motion process in real-time, and force the motion to stop.

Note: During axis motion, it isn't allowed to send new driving instruction to the motion axis, or else it will cancel previous driving and execute next driving instruction.

Although the programming languages are multifarious, they have same nature. In summary, it is three structures and one idea. The three structures are order structure, loop structure and branch structure, which are emphasized in all programming languages, and one idea is the algorithm and module division used to complete design task, which is the important and difficult point of the entire program design.

To ensure the generality, standardized, scalability and easy maintenance, the examples below focus on project design. The examples are divided into the following modules:

motion control module (further package the library function provided by the control card), function realization module (coordinating the code segment of specific process), monitoring module and stop processing module.

Below is the introduction of ADT8860 card function library application in programming languages VB and VC. For other programming languages, please refer to the example of VB and VC.

8.5.1. VB programming example

1.1 PREPARATION

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

(2) Add "adt8860.bas" module in the item according to above method;

1.2 Motion control module

(1) Add a new module in the item and save as "ctrlcard.bas";

(2) In motion control module, first customize the initialization function of the control card and initialize the library function that needs to be packaged in initialization function;

(3) Continue customizing related motion control functions, such as speed setting function, single axis motion function, interpolation motion function, etc.;

(4) The source code of ctrcard.bas follows:

//Content should be added 3-25 ?????

1.3 Function realization module

Interface design

??????

1.4 Monitoring module

//??????

Monitoring module is used to get the driving info of all axes in real-time, display motion info, and control to restricting sending new driving instruction during driving. Monitoring module is completed with timer event, and the code follows:

//????????????

1.5 Stop module

Stop module is used to control the emergencies in the driving process, and the driving of all axes must be stopped immediately. The code of stop module is in the click event of CmdStop button, and the code follows:

//??????????

8.5.2. VC programming example

1.1 Preparation

(1) Create a new item and save as "VCExample.dsw";

(2) Load static library "ADT8860.lib" into the item according to above method;

1.2 Motion control module

(1) Add a new category in the item, save the header file as "CtrlCard.h" and save the source file as "CtrlCard.cpp";

(2) In motion control module, first customize the initialization function of the control card and initialize the library function that needs to be packaged in initialization function;

(3) Continue customizing related motion control functions, such as speed setting function, single axis motion function, interpolation motion function, etc.;

(4) The code of header file "CtrlCard.h" follows:

```
#define    MAXAXIS    6        //maximum axis number
class CCtrlCard
{
    public:
    int Init_Board(int id);
    int Set_Basic_Para(unsigned char port,int mode);
    int Setup_Speed(int id, int axis, int startv, unsigned int speed, unsigned    int add , unsigned
                        int pmm);
    int Setup_SpeedExt(int id, unsigned short *axis, unsigned int    *startv, unsigned int
                        *speed, unsigned    int *add , unsigned    int *pmm);
    int Set_Serial_Init(unsigned char port,unsigned int baud, unsigned char databit, unsigned
                        char stopbit, unsigned char parity);
    int Get_Net_Conn(unsigned char port,int devnum,int conip,unsigned short conport);
    int Close_NetDev(unsigned char port,int devnum);
    int Set_Consle_Port(unsigned char port);
    int Get_Uart_Recv(unsigned char port,unsigned char &rbuff ,int length);
    int Get_Consle_Port(void);
    int Close_Port(unsigned char port);
    int Set_TimeOut(unsigned short time,unsigned char renum);
    int Get_TimeOut(unsigned short *time,unsigned char *renum);
    int Axis_Pmove(int id,int axis, long pulse);
    int MultAxis_Pmove(int id,unsigned short *axis, int *pulse);
    int Interp_Move2(int id,int axis1,    long pulse1,int axis2, long pulse2);
    int Interp_Move3(int id,int axis1,    long pulse1,int axis2, long pulse2,int axis3, long
                        pulse3);
    int Interp_Move4(int id,long pulse1, long pulse2, long pulse3, long pulse4);
    int Interp_Move5(int id,long pulse1, long pulse2, long pulse3, long pulse4, long pulse5);
    int Interp_Move6(int id,long pulse1, long pulse2, long pulse3, long pulse4, long pulse5,
                        long pulse6);
    int StopRun(int id,int mode);
    int Get_Status(int id, unsigned int &value, int mode);
    int Get_RegisterPos(int id,   unsigned int &regpos );
    int Get_ONLY_Status(int id, short int &status1,short int &status2, short int &status3,short
                        int &status4,short   int &status5,short int &status6);
    int Get_CurrentInf(int id,   int *LogPos,   int *ActPos,unsigned   int *Speed );
    int Read_Input(int id,int svbit,int number,unsigned char *status);
    int Write_Output(int id,int svbit,int number,unsigned char *status);
    int Setup_Pos(int id,unsigned short &axis, int &pos, int mode);
    int Setup_PulseMode(int id,int axis, int value,int logic,int dir_logic);
    int Setup_LimitMode(int id,int axis, int value1, int value2, int logic);
    int Setup_Stop0Mode(int id,int axis, int value, int logic);
    int Setup_Stop1Mode(int id,int axis, int value, int logic);
    int Get_OutNum(int id,int svbit,int number,unsigned char &status);
    int Continue_Move( int id,   unsigned short int axis, short int dir);
    int Coord_Mode( int id,   unsigned   int mode);
    int MultAxis_ConfMode( int id, unsigned int conftmode);
    int Command_ExeMode( int id, unsigned int executmode);
    CCtrlCard();
    int Result;             //Return
};
```

(5) The code of source file “CtrlCard.cpp” follows:

```
CCtrlCard::CCtrlCard()
{
}
/****************** Initialization function **********************
 This function includes the library functions commonly used by control card
 initialization, which is the base of calling other functions and therefore must be first
 called in the example program
Return <=0 initialization failed, return >0 initialization successful
*****************************************************/
int CCtrlCard::Init_Board(int id)
{
    Result = pmc_modpara_init() ;           // card initialization function
    if (Result <= 0) return Result;
    for (int i = 1; i<=MAXAXIS; i++)
    {
        pmc_set_limit_mode(id, i, 0, 0, 0);   //set limit mode, set positive/negative limit
        effective, low voltage level effective
        pmc_set_command_pos(id, i, 0);          // clear logical position counter
```

```
            pmc_set_actual_pos (id, i, 0);          // clear actual position counter
            pmc_set_speed (0, i, 50,1000,200,10);// set driving speed
        }
        return 1;
    }

    /*****************************************************
    Function feature: set communication mode
    Input parameter:
    port: corresponding communication port No.
    mode: communication mode of writing data and data type configuration parameter; default
data access mode: readable/writable
    Output parameter:    None
    Return: 0: send successfully, -1: send failed
    *****************************************************/
    int   CCtrlCard::Set_Basic_Para(unsigned char port,int mode)
    {
       Result = pmc_set_signle_para(port, mode);
       return Result;
    }


    /*****************************************************
    Function feature: serial port initialization
    Input parameter: port: serial port (0,1,2,3)
    baud: set the baud rate corresponding to serial port
    databit: serial port data bit (5, 6, 7, 8-bit)
    stopbit: serial port stop bit (1, 2-bit)
     Parity: serial port parity bit (no, odd, even)
    Output parameter:    None
    Return: 0: normal, -1: send failed
    *****************************************************/
    int   CCtrlCard::Set_Serial_Init(unsigned char port,unsigned int baud, unsigned char databit,
           unsigned char stopbit, unsigned char parity)
    {
       Result = pmc_serial_init(port,baud, databit,stopbit,parity,TRUE);
       return Result;
    }
    /*****************************************************
    Function feature: initialize TCP/UDP network connection of Modbus
```

```
Input parameter:
port: network port No., only the port No. for TCP and UDP connection
devnum: device No. corresponding to network communication
conip: connected server IP address
conport: connected server communication port No. (Modbus communication is fixed at 502
generally)
Output parameter:    None
Return: 0: normal, -1: send failed
*******************************************************/
int    CCtrlCard::Get_Net_Conn(unsigned char port,int devnum,int conip,unsigned short
         conport)
{
     Result = pmc_net_conn(port,    devnum, conip,    conport);
     return Result;
}
/*******************************************************
Function feature: close communication port
Input parameter: port: communication port No. (only valid for TCP/UPD network port)
devnum: device No. corresponding to network communication
Output parameter:    None
Return: 0: normal, -1: send failed
*******************************************************/
int    CCtrlCard::Close_NetDev(unsigned char port,int devnum)
{
     Result = pmc_clos_netdev(port, devnum);
     return Result;
}

/******************Set console command communication port *******************
Input parameter: Port: set port No.
Output parameter:    None
Return: 1: true, 0: false
**********************************************************/
int    CCtrlCard::Set_Consle_Port(unsigned char port)
{
     Result = pmc_set_consleport(port);
     return Result;
}

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

```
Function feature: serial port read data function
Input parameter:
Port: communication port No.
rbuff: storage area of read data
length: specify the length of read data
Output parameter:    None
Return: actually received data length
**********************************************************/
int   CCtrlCard::Get_Uart_Recv(unsigned char port,unsigned char &rbuff ,int length)
{
    Result = pmc_uart_recv(port, &rbuff,length);
    return Result;
}

/*****************Get console command communication port ******************
Input parameter: None
Output parameter:    None
Return: Set console command communication port No.
*****************************************************************/
int   CCtrlCard::Get_Consle_Port(void)
{
    Result = pmc_get_consleport();
    return Result;
}

/*****************************************************************
Function name: ModbusClosePort()
Description: close corresponding connection port of Modbus
Input parameter:
Port: communication port No. (including serial port and network port)
Output parameter:    None
Return: 0: normal, -1: send failed
******************************************************************/

int   CCtrlCard::Close_Port(unsigned char port)
{
    Result = pmc_close_port(port);

    return Result;
}
```

```
/************************************************************
Function feature: get timeout and repeat times of corresponding port
Input parameter:
*timeouts: timeout (unit: ms)
*repeat_times: repeat times
Output parameter:    None
Return: TRUE: normal
************************************************************/
int    CCtrlCard::Get_TimeOut(unsigned short *time,unsigned char *renum)
{
    Result = pmc_get_timeout(time,renum);

    return Result;
}

/***********************************************************
Function feature: set timeout and repeat times of corresponding port
Input parameter:
time: timeout (unit:ms) > 500ms
renum:   repeat times
Output parameter:    None
Return: TRUE:  normal
***********************************************************/
int    CCtrlCard::Set_TimeOut(unsigned short time,unsigned char renum)
{
    Result = pmc_set_timeout(time,renum);
    return Result;
}

/***********************************************************
Function feature: set starting speed, driving speed and acceleration of the axis
Input parameter:
Id: slave station No.
Axis: axis number
Startv:  starting speed
Speed: driving speed
Add: acceleration
Pmm: pulse equivalent numerator
Return: 0: true,  1: false
```

```
*************************************************************/
int CCtrlCard::Setup_Speed(int id, int axis, int startv, unsigned int speed, unsigned  int add ,
      unsigned  int pmm)
{
    if (startv - speed >= 0) //constant speed motion
    {
        Result = pmc_set_speed(id,axis,startv,startv,add,pmm);
    }
    else                         //acceleration/deceleration motion
    {
        Result = pmc_set_speed(id,axis,startv,speed,add,pmm);
    }
    return Result;
}

/*********************************************************
Function feature: set starting speed, driving speed and acceleration of multi-axis
Parameter:
Id: slave station No.
Axis: axis number
startv:  starting speed
speed: driving speed
add: acceleration
pmm: pulse equivalent numerator
Return: 0: true,  1: false
*************************************************************/
int CCtrlCard::Setup_SpeedExt(int id,  unsigned short *axis, unsigned int *startv, unsigned
      int *speed, unsigned  int *add , unsigned  int *pmm)
{
    Result = pmc_set_speedExt(id,axis,startv,speed,add,pmm,6);
    return Result;
}

/******************** Single axis driving function *********************
Function feature: single axis quantitative driving function
Parameter:
Id: slave station No.
Axis: axis number
Value: output pulses
Return: 0: true,  1: false
```

```
*****************************************************/
int CCtrlCard::Axis_Pmove(int id,int axis, long pulse)
{
    Result = pmc_pmove(id, axis, pulse);
    return Result;
}

/*****************************************************
Function feature: this function is used to drive multi-axis motion
Parameter:
Id: slave station No.
Value: output pulses
Return=0: true, Return=1: false
*****************************************************/
int CCtrlCard::MultAxis_Pmove(int id,unsigned short *axis, int *pulse)
{
    Result = pmc_pmoveExt(id, axis, pulse,6);
    return Result;
}

/*****************************************************
Function feature: random two axes interpolation motion
Parameter:
Id: slave station No.
axis1, axis2: axis number 1-6
pulse1, pulse2: pulses
Return: 0: true,  1: false
*****************************************************/
int CCtrlCard::Interp_Move2(int id,int axis1,   long pulse1,int axis2, long pulse2)
{
    Result = pmc_line2(id, axis1, pulse1,axis2, pulse2);
    return Result;
}

/*****************************************************
Function feature: random three axes interpolation motion
Parameter:
Id: slave station No.
axis1, axis2, axis3: axis number
pulse1, pulse2, pulse3: pulses
```

```
Return: 0: true,  1: false
*******************************************************/
int CCtrlCard::Interp_Move3(int id,int axis1,   long pulse1,int axis2, long pulse2,int axis3,
      long pulse3)
{
    Result = pmc_line3(id, axis1, pulse1,axis2,pulse2, axis3, pulse3);
    return Result;
}

/*********************************************************
Function feature: first four axes interpolation motion
Parameter:
Id: slave station No.
pulse1, pulse2, pulse3, pulse4: pulses
Return: 0: true,  1: false
*********************************************************/
int CCtrlCard::Interp_Move4(int id,long pulse1, long pulse2, long pulse3, long pulse4)
{
    Result = pmc_line4(id, pulse1, pulse2, pulse3, pulse4);
    return Result;
}

/*********************************************************
Function feature: drive first five axes interpolation motion


Parameter:
Id: slave station No.
pulse1,pulse2,pulse3,pulse4,pulse5: pulses
Return: 0: true,  1: false
*********************************************************/
int CCtrlCard::Interp_Move5(int id,long pulse1, long pulse2, long pulse3, long pulse4, long
      pulse5)
{
    Result = pmc_line5(id, pulse1, pulse2, pulse3, pulse4, pulse5);
    return Result;
}

/*********************************************************
Function feature: six axes interpolation motion
```

```
Parameter:
Id: slave station No.
pulse1, pulse2, pulse3, pulse4, pulse5, pulse6: pulses
Return: 0: true,  1: false
**********************************************************/
int CCtrlCard::Interp_Move6(int id,long pulse1, long pulse2, long pulse3, long pulse4, long
      pulse5, long pulse6)
{
    Result = pmc_line6(id, pulse1, pulse2, pulse3, pulse4, pulse5, pulse6);
    return Result;
}

/**********************************************************
Function feature: this function provides immediate stop mode and decelerating stop mode
Parameter:
Id: slave station No.
mode: 0-immediate stop, 1-pause, 2-resume
Return: 0: true,  1: false
**********************************************************/
int CCtrlCard::StopRun(int id,int mode )
{
    if(mode==0)
    {
         Result = pmc_stop(id);
    }
    else if(mode==1)
    {
         Result = pmc_pause(id);
    }
    else if(mode==2)
    {
         Result = pmc_resume(id);
    }
      return Result;
}

/**********************************************************
Function feature: this function is used to get the driving state of single axis
Parameter:
Id: slave station No.
```

```
status1, status2, status3, status4, status5, status6: status indicator (0- driving ends, non-0 -
driving)
Return: 0: true,  1: false
**********************************************************/
int CCtrlCard::Get_ONLY_Status(int id, short int &status1,short int &status2, short int
      &status3,short int &status4,short   int &status5,short int &status6)
{
    Result=pmc_get_axis_status(id,&status1,&status2,&status3,&status4,&status5,&status6);

    return Result;
}

/**********************************************************
Function feature: this function is used to get the driving state or interpolation driving state of
the axis
Parameter:
Id: slave station No.
Value: status indicator (0- driving ends, non-0 - driving)
Mode: 0- get interpolation driving state, 1- get home driving state, 2- get system driving state
Return: 0: true,  1: false
**********************************************************/
int CCtrlCard::Get_Status(int id, unsigned int &value, int mode)
{
    if (mode==0)                // get interpolation driving state
    {
        Result=pmc_get_inp_status(id,&value);
    }
    else   if (mode==1)     // get home driving state
    {
        Result=pmc_get_back_status(id,&value);
    }
      else if(mode==2)           // get system driving state
    {
        Result=pmc_get_run_status(id,&value);
    }
    return Result;
}

/**********************************************************
Function feature: get or set axis logic position of the coordinate system specified in D4 register
Input parameter:
```

```
id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission
regpos: axis logic position of the coordinate system specified in the register
Return: 0: send successfully, -1: send failed
***********************************************************/
int CCtrlCard::Get_RegisterPos(int id,    unsigned int &regpos )
{
    Result = pmc_get_register_pos(id, &regpos);
    return Result;
}

/***********************************************************
Function feature: used to feed back current logic position, actual position and running speed of the axis
Parameter:
Axis: axis number
LogPos: logic position
ActPos: actual position
Speed: operating speed
Return: 0: true,  1: false
***********************************************************/

int CCtrlCard::Get_CurrentInf(int id,    int *LogPos, int *ActPos, unsigned int *Speed )
{
    Result = pmc_get_command_pos(id, LogPos,ActPos,Speed);
    return Result;
}

/***********************************************************
Function feature: read single input point
Parameter:
id: slave station No.
svbit: input port No.
number: continuously read input port number
status:  number of saving input status, 0 – low voltage level, 1 – high voltage level
Return: 0: true, -1: false
************************************************************/
int CCtrlCard::Read_Input(int id,int svbit,int number,unsigned char *status)
{
    Result = pmc_read_Inbit(id,svbit, number,status);
    return Result;
}

/***********************************************************
Function feature: output signal of single point
Parameter:
id: slave station No.
svbit: input port No.
number: continuously read input port number
status:  save the array of input status     0- low, 1- high
Return: 0: true,  1: false
***********************************************************/
int CCtrlCard::Write_Output(int id,int svbit,int number,unsigned char *status)
{
    Result = pmc_write_Outbit(id,svbit, number,status);
    return Result;
}

/************************************************************
Function feature: set logic position and actual position
Parameter:
id: slave station No.
axis: axis number
pos: set position value
mode: 0- set logic position, non-0- set actual position
Return: 0: true,  1: false
************************************************************/

int CCtrlCard::Setup_Pos(int id,unsigned short &axis, int &pos, int mode)
{
    if(mode==0)
    {
        Result = pmc_set_command_posExt(id, &axis, &pos,6);
    }
    else
    {
        Result = pmc_set_actual_posExt(id, &axis, &pos,6);
    }
    return Result;
}

/********************************************************
Function feature: set the working mode of pulse
Parameter   id: slave station No.
axis: axis number (1 – 6)
value: 0- pulse + pulse          1- pulse + direction
logic: 0- positive logic pulse              1- negative logic pulse
dir-logic: 0- direction signal positive logic       1- direction signal negative logic
Return: 0: send successfully, -1: send failed
*********************************************************/
int CCtrlCard::Setup_PulseMode(int id,int axis, int value,int logic,int dir_logic)
{
    Result = pmc_set_pulse_mode(id, axis, value, logic, dir_logic);
    return Result;
}

/********************************************************
Function feature: set the mode of positive/negative limit input nLMT signal
Parameter:
id: slave station No.
axis: axis number
value1: 0- positive limit valid      1- positive limit invalid
value2 0- negative limit valid    1- negative limit invalid
logic 0 - low voltage level effective    1 - high voltage level effective
Default mode: positive limit valid, negative limit valid, low voltage level effective
Return: 0: true,  1: false
*********************************************************/
int CCtrlCard::Setup_LimitMode(int id,int axis, int value1, int value2, int logic)
{
    Result = pmc_set_limit_mode(id, axis, value1, value2, logic);
    return Result;
}
/*******************************************************
Function feature: set the mode of stop0 signal
Parameter:
id: slave station No.
Axis: axis number
```

```
Value: 0 - invalid          1 - valid
Logic: 0 - low voltage level effective 1 - high voltage level effective
Default mode: invalid
Return: 0: true,  1: false
*******************************************************/
int CCtrlCard::Setup_Stop0Mode(int id,int axis, int value, int logic)
{
    Result = pmc_set_stop0_mode(id, axis, value ,logic);
    return Result;
}

/*******************************************************
Function feature: set the mode of stop1 signal
Parameter:
id: slave station No.
axis: axis number
value: 0- invalid   1 - valid
logic: 0 - low voltage level effective   1 - high voltage level effective
Default mode: invalid
Return: 0: true,  1: false
   *******************************************************/
int CCtrlCard::Setup_Stop1Mode(int id,int axis, int value, int logic)
{
    Result = pmc_set_stop1_mode(id, axis, value ,logic);

    return Result;
}

/****************Get output point ******************************
Function feature: get output point
Parameter:
id: slave station No.
svbit: input port No.
number: continuously read input port number
status:  save the array of input status     0- low, 1- high
Return: specify the current state of the port, -1: parameter error
*******************************************************/
int CCtrlCard::Get_OutNum(int id,int svbit,int number,unsigned char &status)
{
        Result=pmc_read_Outbit( id, svbit, number,&status);
        return Result;
    }

    /****************************************
     Function feature: continuous driving of every axis
    Parameter:
     id: slave station No.; invalid for TCP/UDP transmission; connection address and port No.
           should be set for UDP transmission
    axis: axis number (1 - 6)
    dir: running direction, 0: positive, non-zero: negative
    Output parameter:    None
    Return: 0: send successfully, -1: send failed
    ******************************************************/
    int    CCtrlCard::Continue_Move( int id,    unsigned short int axis, short int dir)
    {
        Result=pmc_continue_move( id, axis, dir);
        return Result;
    }

    /***********************************************************
    Function feature: set current coordinate mode of current task
    Parameter:
    id: slave station No.; invalid for TCP/UDP transmission; connection address and port No.
should be set for UDP transmission
    mode: 0: increment coordinate mode, 1: absolute coordinate mode
    Return: 0: send successfully, -1: send failed
    **********************************************************/
    int    CCtrlCard::Coord_Mode( int id, unsigned int mode)
    {
        Result=pmc_coord_mode( id, mode);
        return Result;
    }

    /**********************************************************
    Function feature: set multitask axis motion conflict treatment mode
    Input parameter:
    id: slave station No.; invalid for TCP/UDP transmission; connection address and port No.
should be set for UDP transmission
    conftmode: 0: pause and report error
                    1: execute new motion when the system is in idle,
                    2: if there is no axis conflict in non-interpolation state, allow executing
multi-axis linkage again
```

Return: 0: send successfully, -1: send failed

```
*************************************************************/
int   CCtrlCard::MultAxis_ConfMode( int id, unsigned int conftmode)
{
    Result=pmc_multaxis_conftmode( id, conftmode);
    return Result;
}
```

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

Function feature: motion instruction execution mode

Parameter:

id: slave station No.; invalid for TCP/UDP transmission; connection address and port No. should be set for UDP transmission

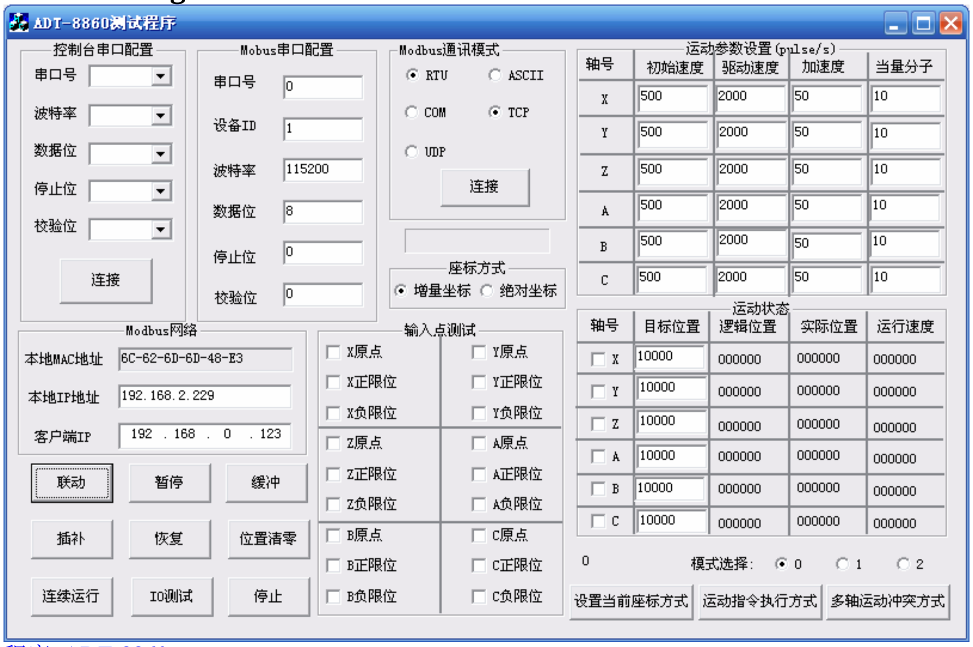
executmode: 0 - execute next motion command only after execution of the current motion command in real-time, and each motion command has independent acceleration and deceleration segments;
1 - send continuous motion commands to hardware cache, and then the hardware executes continuous interpolation motion; each motion command is uniform motion, and doesn't have acceleration and deceleration segments;
2 - implement speed forward smoothing for continuous motion command according to its motion track; default: 0

Return: 0: send successfully, -1: send failed

```
***********************************************************/
int   CCtrlCard::Command_ExeMode( int id, unsigned int executmode)
{
    Result=pmc_command_executmode( id, executmode);
    return Result;
}
```

1.3 Function realization module

1.3.1 Interface design



Mobus串口配置

串口号	0
设备ID	1
波特率	115200
数据位	8
停止位	0
校验位	0

Modbus通讯模式: [✓ RTU] ASCII / COM [✓ TCP] / UDP

座标方式: [✓ 增量坐标] 绝对坐标

Modbus网络

本地MAC地址	6C-62-6D-6D-48-E3
本地IP地址	192.168.2.229
客户端IP	192 . 168 . 0 . 123

运动参数设置(pulse/s)

轴号	初始速度	驱动速度	加速度	当量分子
X	500	2000	50	10
Y	500	2000	50	10
Z	500	2000	50	10
A	500	2000	50	10
B	500	2000	50	10
C	500	2000	50	10

运动状态

轴号	目标位置	逻辑位置	实际位置	运行速度
X	10000	000000	000000	000000
Y	10000	000000	000000	000000
Z	10000	000000	000000	000000
A	10000	000000	000000	000000
B	10000	000000	000000	000000
C	10000	000000	000000	000000

0 模式选择: [✓ 0] 1 2

ADT-8860 测试程序 ADT-8860 test program
串口号 Serial port No. / 波特率 Baud rate / 数据位置 Data bit / 停止位 / Stop bit / 检验位 Parity
串口号 Serial port No. / 设备 ID Device ID / 波特率 Baud rate / 数据位 Data bit / 停止位 Stop bit / 检验位 Parity
连接 Connect Modbus 网络 Modbus network

本地 MAC 地址 MAC address / 本地 IP 地址 IP address / 客户端 IP Client IP
联动 Linkage \ 暂停 Pause \ 缓冲 Buffer 插补 Interpolate \ 恢复 Resume \位置清零 Clear position
连续运行 Continuous running \ IO 测试 IO test \ 停止 Stop
座标方式 Coordinate mode 增量坐标 Increment \ 绝对坐标 Absolute
原点 home / 正限位 positive limit / 负限位 negative limit
运动参数设置（pules/s）Motion parameter setting (pulse/s)
轴号 Axis No. \ 初始速度 Starting speed \ 驱动速度 Driving speed \ 加速度 Acceleration \ 当量分子 Equivalent numerator
运动状态 Motion status / 轴号 Axis No. \ 目标位置 Target position \ 逻辑位置 Logic position \ 实际位置 Actual position \ 运行速度 Running speed
模式选择 Mode selection / 设置当前座标方式 Set current coordinate mode \ 运动指令执行方式 Motion instruction execution mode \ 多轴运动冲突方式 Multi-axis motion conflict mode

Note:

(1) Speed setting—used to set the start speed, driving speed and acceleration of every axis; position setting—set the driving pulse of every axis; driving info—display the logical position, actual position and running speed of every axis in real-time.

(2) Driving object—confirm the axis for linkage or interpolation by selecting driving object;

(3) Linkage—send single axis driving instruction to all axes of selected driving object; interpolation—send interpolation instruction to all axes of selected driving object; stop—stop the pulse output of all axes;

All above data use pulse as the unit.

1.3.2 Initialization code of motion control card

```
int ret;
ret=g_CtrlCard.Init_Board(m_iNoid);
if(!ret)
{
    AfxMessageBox("Motion control card initialization succeeded!");
}
else
{
    AfxMessageBox("Motion control card initialization failed!");
}
```

1.3.3 Monitoring module

Monitoring module is used to get the driving info of all axes in real-time, display motion info, and control to restricting sending new driving instruction during driving. Monitoring module is completed with timer event, and the code follows:

```
void CMy8860DemoDlg::OnTimer(UINT nIDEvent)
{
    CString str;
    CStatic *lbl;
    int logpos[6];
    int  actpos[6];
    UINT speedpos[6];

    UINT
    nID1[]={IDC_LOG_POSX,IDC_LOG_POSY,IDC_LOG_POSZ,IDC_LOG_POSA,IDC_
    LOG_POSB,                                    IDC_LOG_POSC};
```

```
UINT
nID2[]={IDC_ACT_POSX,IDC_ACT_POSY,IDC_ACT_POSZ,IDC_ACT_POSA,IDC_
ACT_POSB,IDC_ACT_POSC};
UINT
nID3[]={IDC_RUNSPEED_X,IDC_RUNSPEED_Y,IDC_RUNSPEED_Z,IDC_RUNSPE
ED_A,IDC_RUNSPEED_B,IDC_RUNSPEED_C};

int ret=g_CtrlCard.Get_CurrentInf(m_iNoid,logpos,actpos,speedpos);
if(ret == 0)
{
    for (int i=1; i<7; i++)
    {
        ::DoEvent();
        lbl=(CStatic*)GetDlgItem(nID1[i-1]);
        str.Format("%ld",logpos[i-1]);
        lbl->SetWindowText(str);

        lbl=(CStatic*)GetDlgItem(nID2[i-1]);
        str.Format("%ld",actpos[i-1]);
        lbl->SetWindowText(str);

        lbl=(CStatic*)GetDlgItem(nID3[i-1]);
        str.Format("%ld",speedpos[i-1]);
        lbl->SetWindowText(str);
    }

}

UINT
  nIDIN5[12]={IDC_HOME_X,IDC_HOME_Y,IDC_HOME_Z,IDC_HOME_A,
    IDC_PLIMT_X,IDC_NLIMT_X,IDC_PLIMT_Y,IDC_NLIMT_Y,IDC_PLIMT_Z,
    IDC_NLIMT_Z,IDC_PLIMT_A,IDC_NLIMT_A    };

UINT
nIDIN2[6]={IDC_HOME_B,IDC_HOME_C,IDC_PLIMT_B,IDC_NLIMT_B,IDC_PLIM
T_C,IDC_NLIMT_C};

INT16U bit_start=0;
INT16U bit_num=85;
INT8U status3[86];
int ii=0;
```

```
CButton *btn;
CButton *btn2;
int ret2;
ret2=g_CtrlCard.Read_Input(m_iNoid, bit_start, bit_num, status3);
if(ret2 == 0)
{
    for( ii=0;ii<12;ii++)
    {
        if(status3[ii]==0)
        {
            btn=(CButton*)GetDlgItem(nIDIN5[ii]);
            btn->SetCheck(1);
        }
        else
        {
            btn=(CButton*)GetDlgItem(nIDIN5[ii]);

            btn->SetCheck(0);
        }
    }

    for(ii=0;ii<6;ii++)
    {
        btn2=(CButton*)GetDlgItem(nIDIN2[ii]);

        if(status3[ii+17]==0)
        {
            btn2->SetCheck(1);
        }
        else
        {
            btn2->SetCheck(0);
        }
    }
}
INT32U regpos=0;
g_CtrlCard.Get_RegisterPos(m_iNoid,regpos);
lbl=(CStatic*)GetDlgItem(IDC_REG_POS);
str.Format("%ld",regpos);
lbl->SetWindowText(str);
```

```
    CDialog::OnTimer(nIDEvent);
 }
```

1.3.4 Stop module

Stop module is used to control the emergencies in the driving process, and the driving of all axes must be stopped immediately. The code of stop module is in the click event of CmdStop button, and the code follows:

```
void CMy8860DemoDlg::OnButtonStop()
{
    g_CtrlCard.StopRun(m_iNoid,0);
}
```

9 Network Settings

9.1 Network environment

ADT-8860 control card can run in the LAN that consists of several subnets, but it is recommended to run in dedicated subnet. The subnet shouldn't be accessed by irrespective network to avoid communication delay or failure due to busy network. When the host controls in real-time, please close other applications to ensure that the host has a better response performance.
Since the client LAN may have different environments, we can't ensure that the IP address and NIC address (MAC) of the controller device are unique in client network before the device is delivered. The controller only has one default IP address and NIC address (MAC) before delivery, and therefore all new control cards should be confirmed with serial testing tool SSCOM32.EXE (supplied) or the network option should be re-configured, and ensure that the IP address and NIC address are unique in the LAN.

Default network configuration follows:

IP address:	192.168.0.123
Subnet mask:	255.255.255.0
Default gateway:	192.168.0.1
NIC address (MAC):	0B-16-21-2C-37-42

9.2 Configuring network parameters

The command to configure IP address, subnet mask, and default gateway is **SET_IP n "xxx.xxx.xxx.xxx",** and the range of n is 0-2, indicating the objects respectively.
The command to configure MAC address is **SET_MAC "xx-xx-xx-xx-xx-xx"**
After setting the network parameters, please save the system parameters and restart the system to take effect.

9.3 Network configuration related information

IP address: 4-section decimal value, each section is separated with '.', the value in every section shouldn't be larger than 255, and the first section shouldn't be larger than or equal to 223 or equal to 1. Generally, the first three sections are subnet number, and the last section is the host number in the subnet (there is also exception, but only the general condition is described). The IP address of control card should be in the same subnet with the control host. If the control card and the control host are in different subnets, please adjust the mask properly (it is recommended to set the control card and the control host in the same subnet).

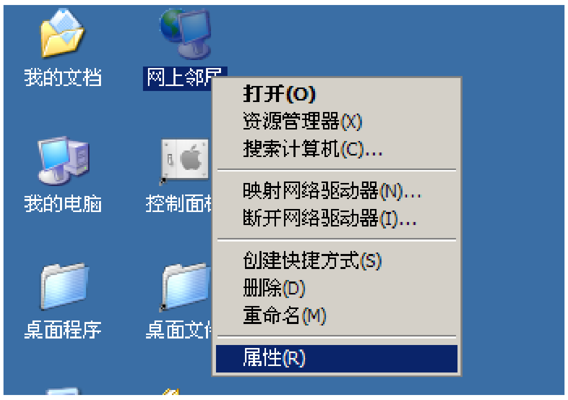
Subnet mask: 4-section decimal value, each section is separated with '.', the value in every section shouldn't be larger than 255, and is used to distinguish the subnet number and host number.

Default gateway: 4-section decimal value, each section is separated with '.', the value in every section shouldn't be larger than 255, and the first section shouldn't be larger than or equal to 223 or equal to 1. If there is real gateway in the subnet, please enter the real IP address of the gateway, or else set the default gateway to the first host number in the subnet, e.g. xxx.xxx.xxx.1.

NIC address (MAC): 6-section hexadecimal value, each section is separated with '-', the value in every section shouldn't be larger than FF, and the first section shouldn't be equal to 01.

9.4 Host (upper PC) network configuration

The network configuration of control host should be modified in TCP/IP Protocol Properties dialog box. The method to open the TCP/IP Protocol Properties dialog box in Windows XP follows:
In the Start menu or on the desktop, find My Network Places -> right click My Network Places and click Properties -> open the Network Connections window -> right click Local Area Connection and click Properties -> open Local Area Connection Properties window -> activate General tab -> select Internet Protocol (TCP/IP) -> click Properties to open the Internet Protocol (TCP/IP) Properties window, as shown below.



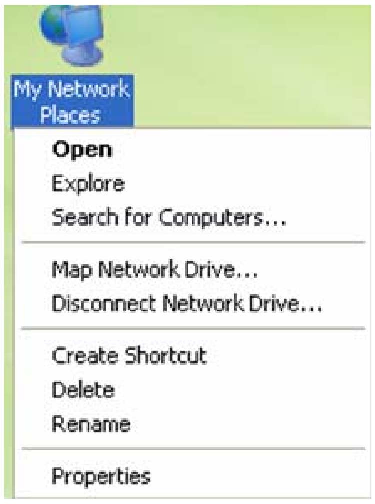
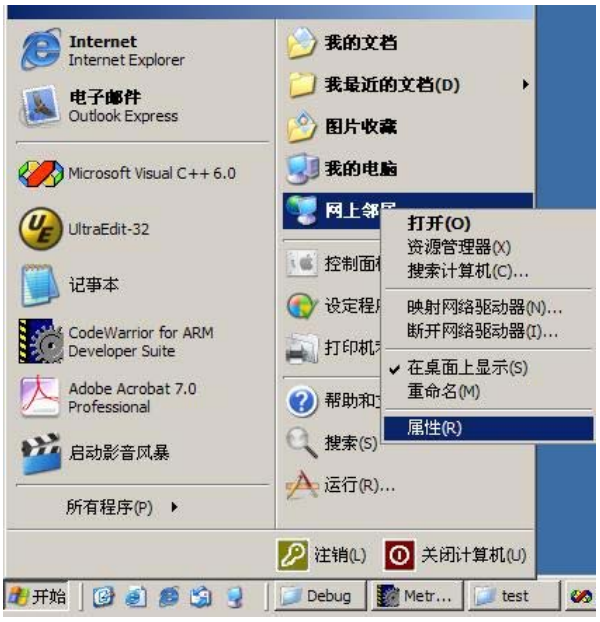


Fig. 7



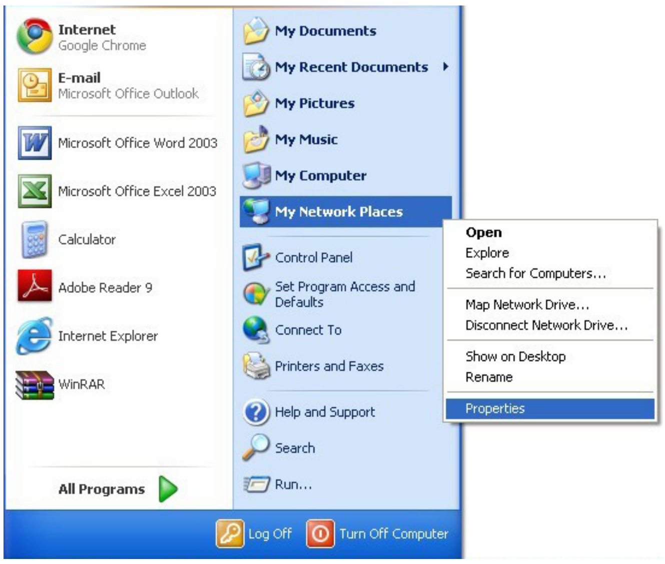
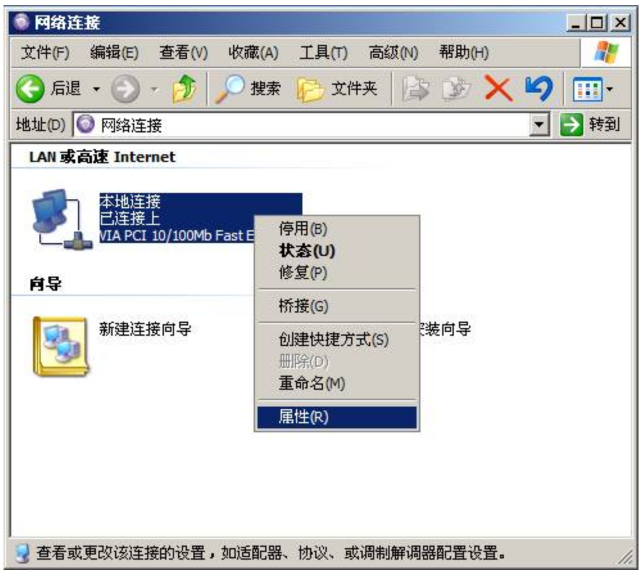


Fig. 8



查看或更改该连接的设置，如适配器、协议、或调制解调器配置设置。

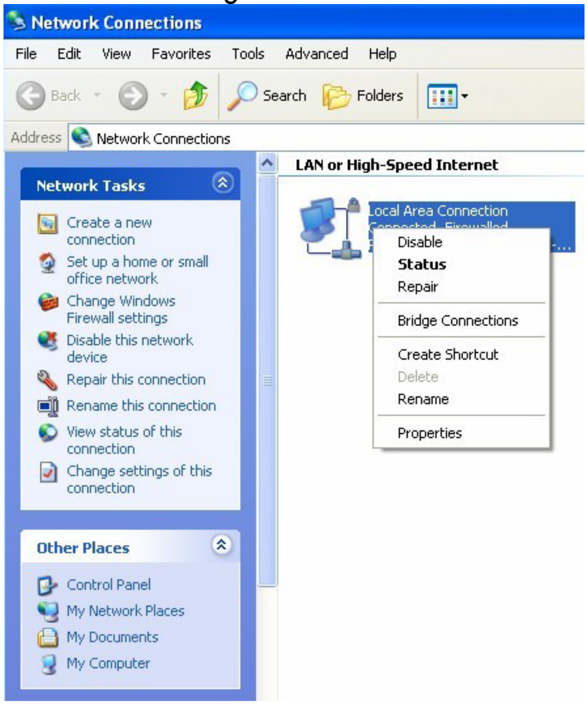
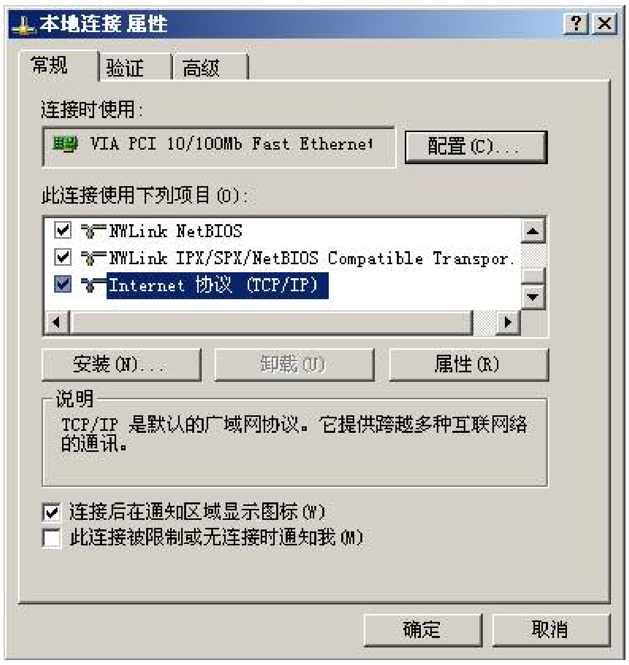


Fig. 9



连接时使用：

VIA PCI 10/100Mb Fast Ethernet

此连接使用下列项目(O)：

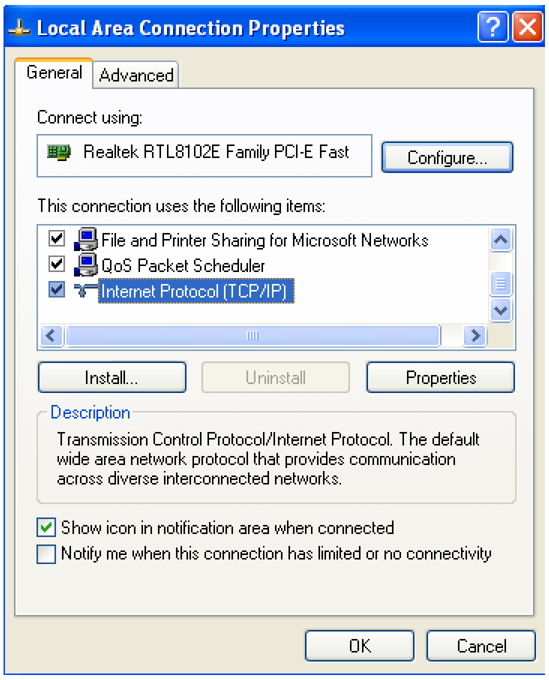
- ☑ NWLink NetBIOS
- ☑ NWLink IPX/SPX/NetBIOS Compatible Transpor
- ☑ Internet 协议 (TCP/IP)

说明

TCP/IP 是默认的广域网协议。它提供跨越多种互联网络的通讯。

☑ 连接后在通知区域显示图标(W)

☐ 此连接被限制或无连接时通知我(M)


Local Area Connection Properties
General Advanced
Configure...
Install... Uninstall Properties
OK Cancel

Connect using:

Realtek RTL8102E Family PCI-E Fast

This connection uses the following items:

- ☑ File and Printer Sharing for Microsoft Networks
- ☑ QoS Packet Scheduler
- ☑ Internet Protocol (TCP/IP)

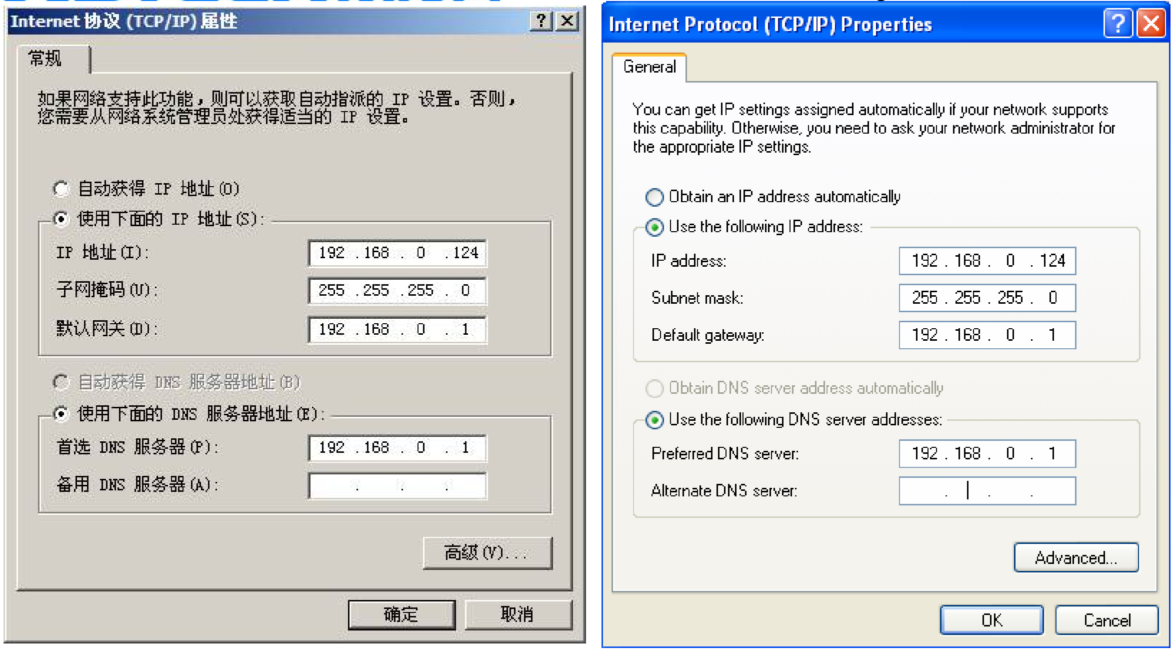
Description

Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks.

☑ Show icon in notification area when connected

☐ Notify me when this connection has limited or no connectivity

Fig. 10



Internet 协议 (TCP/IP) 属性

常规

如果网络支持此功能，则可以获取自动指派的 IP 设置。否则，您需要从网络系统管理员处获得适当的 IP 设置。

- 自动获得 IP 地址(O)
- [✓ 使用下面的 IP 地址(S):]

IP 地址(I):	192 . 168 . 0 . 124
子网掩码(U):	255 . 255 . 255 . 0
默认网关(D):	192 . 168 . 0 . 1

- 自动获得 DNS 服务器地址(B)
- [✓ 使用下面的 DNS 服务器地址(E):]

首选 DNS 服务器(P):	192 . 168 . 0 . 1
备用 DNS 服务器(A):	. . .

高级(V)...

确定 取消

Internet Protocol (TCP/IP) Properties

General

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

- Obtain an IP address automatically
- [✓ Use the following IP address:]

IP address:	192 . 168 . 0 . 124
Subnet mask:	255 . 255 . 255 . 0
Default gateway:	192 . 168 . 0 . 1

- Obtain DNS server address automatically
- [✓ Use the following DNS server addresses:]

Preferred DNS server:	192 . 168 . 0 . 1
Alternate DNS server:	. . .

Advanced...

OK Cancel

Fig. 11

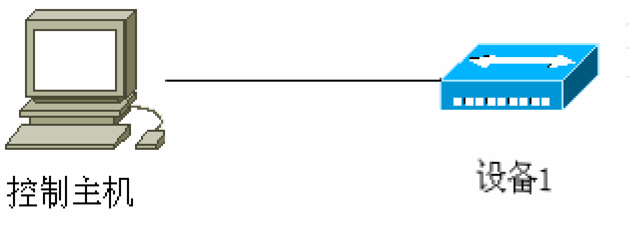
In Internet Protocol (TCP/IP) Properties window, select "Use the following IP address", and then the user can set the IP address, Subnet mask and Default gateway of the control host according to the subnet number in the LAN. If the subnet doesn't have real gateway, set the gateway to the first host number in the subnet, e.g.: 192.168.0.1.

In actual application, the subnet number of the control card should be consistent with the subnet number of control host. If the subnet number of the control card is inconsistent with the subnet number of control host, please modify the network configuration file of the control card through serial software tool. If the control card and the control host are connected directly through crossover cable, it is also possible to adapt to the control card by modifying the IP address of the control host.

9.5 Network topology structure

■ Point to point interconnection

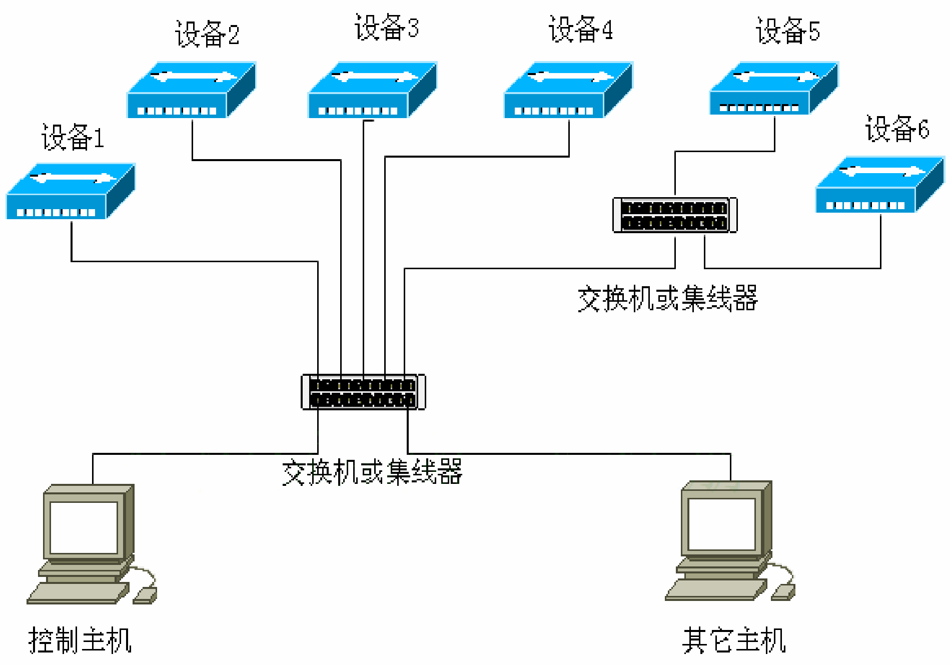
Control host and the device are connected directly through 568B crossover cable



控制主机 Control host　　设备 1 Device 1

■ Star topology interconnection

Control host and the device are connected through switch or hub with 568A straight-through cable. While building network, the industrial Ethernet standard is recommended.



设备 1 Device 1 /设备 2 Device 2 / 设备 3 Device 3 / 设备 4 Device 4 / 设备 5 Device 5 / 设备 61 Device 6
交换机或集线器 Switch or hub 控制主机 Control host 其它主机 Other host

9.6 Common network failures

1) The IP address and NIC address of the control card aren't reconfigured, causing address conflict. Solution: reset the network options of the control card.

2) The IP address of the control card and the IP address of the control host aren't in same subnet. Solution: set the IP addresses of control host and control card in the same subnet, or modify the subnet mask.

3) Several devices with same IP address and NIC address exist in the LAN. The solution is same to "1)"

4) The network cable doesn't match; the control card and the control host should be connected with crossover cable directly, while the control card and HUB or switch can be connected with either crossover cable or straight-through cable.

5) The IP address and MAC address of the control card specified during network connection do not exist. Solution: get (or modify) real control card IP address and MAC address with serial tool, and then confirm the uniqueness of the address in the subnet.

6) The physical connection isn't reliable; please check.

7) Default gateway setting isn't appropriate.

8) If above methods aren't effective, please cut off the power supply and restart the control card.

10 System Alarm Info

All system alarms can be shown in the console, including alarm source, alarm ID, alarm code, code position, and alarm reason, and write these data into the register area started with C96 register.

Alarm source

This is the task program that the alarm originates, and the task number (0-31) is the alarm source. If the error originates from the console, the alarm source will be 32.

Alarm ID

Every type of alarm corresponds to an alarm ID so as to confirm the alarm type according to alarm ID table.

Alarm code

If the alarm reason originates from console or program code keywords, the console will record the error code.

Code position

If the alarm reason originates from console or program code keywords, the console will record the offset position of the error code in the program.

Alarm reason

Simple description of system alarm reason

10.1 PMC program execution alarm

000	No error
001	Keyword error
002	Motion axis conflict error
003	Axis No. error
004	Logic axis isn’t configured
005	Modbus command axis No. error
006	Motion end point position and target position error overtravel
007	Arc calculation error
008	Arc parameter error
009	Limit alarm error
010	Speed is zero
011	Variable name error
012	Parameter error

013	Bit address overtravel error
014	Register address overtravel error
015	Workpiece coordinate system doesn’t exist
016	New workpiece coordinate system repeats
017	Subroutine label error
018	Block No. error
019	Parameter file operation error
020	File name error
021	File doesn’t exist
022	Coordinate system clear error
023	Serial parameter setting error
024	IP address parameter error
025	MAC address parameter error
026	Speed mode setting error
027	Four bytes of Modbus access address aren’t aligned
028	Task doesn’t exist
029	Task already exists
030	Program loading error
031	Program file doesn’t exist
032	Task is running and can’t execute other conflicting actions
033	Program isn’t loaded
034	Task No. illegal overtravel error
035	Task uninstallation error
036	Task isn’t run

11 Precautions and troubleshooting

Precautions

Safety precautions:

1. Do not open the enclosure without permission.
2. Please cut off the power supply if the card won't be used.
3. Do not let liquid, dust or ferrous powder enter the card.

Precautions for proper operation:

Improper operation will cause abnormal working, and even damage the card; therefore, please follow the precautions below to operate the offline card properly.

1. If the output relay is non-solid-state relay, please connect a freewheeling diode to the relay coil in parallel. Check whether the power supply complies with the requirement to avoid burning the card.
2. Card life depends on the ambient temperature. If the temperature on the processing site is too high, please install a cooling fan.
3. Avoid operating in the environment with high temperature, moisture, dust or corrosive gas.
4. Add a rubber pad for buffering in the places with strong vibration.
5. The power supply of offline card is 24VDC.
6. The voltage of output circuit is 12VDC ~ 24VDC and 24VDC is recommended.

Maintenance

Maintenance precautions

1. Before maintaining the offline card, please cut off the power supply of the main loop.
2. The operator should make sure that the power supply has been cut off to avoid accidents.

Startup precautions

Before connecting to the power supply, check whether the wires is connected properly, and pay attention to the influence of strong current and weak current while wiring (please refer to the circuit diagram of the control box for details).

1. Check whether the current of the drive is proper, and whether the subdivision setting is appropriate
2. Check whether the motor and corresponding motion axis are proper
3. Check the correlation of input and output circuits
4. Check whether the power switch of the control box is disconnected

Inspection item and period

Under general conditions (environment: daily average temperature: 30℃, load rate: 80%, running rate 12h/day), please implement daily inspection and periodic inspection according to the items below.

Daily inspection	Daily	● Check the ambient temperature and dust ● Check whether there is abnormal vibration and sound ● Check whether the vents are blocked by yarn

Troubleshooting

- **Motion axis XYZA doesn't act**

1) Check whether XYZA output mode, direction, pulses and millimeter per rotation in [Basic Parameter Settings] are appropriate.

2) Check whether XYZA has limit input signal in [I/O Testing]; if yes, check the wiring

3) Check whether the motor and the drive are connected properly, whether the drive and the card are connected properly, and whether the drive provides sufficient current to drive the motor; for servo, check whether the control mode of servo is set properly (this software supports position control mode)

4) Check whether the motor load of XYZA is too high

- **Motion axis XYZA screaming**

1) Check whether the speed of XYZA is too high; generally, the step motor should run at 5-6 rotations per second when there is no load, and servo should be higher.

2) Check whether the current provided by the drive is sufficient or too high.

- **The processed graphs do not have accurate size and the position has error**.

1) Check whether motion axis XYZU has gap or belt toughness error.

2) Check whether the pulses and millimeter per rotation of motion axis XYZU are accurate.